

# IMSL™

## JMSL™ Numerical Library Chart Programmer's Guide



IMSL Technology for Java™ Programmers

Trusted For Over **30** Years

**Visual Numerics, Inc. – United States**

Corporate Headquarters  
 2000 Crow Canyon Place, Suite 270  
 San Ramon, CA 94583  
 PHONE: 925.807.0138  
 FAX: 925.807.0145  
 e-mail: [info@vni.com](mailto:info@vni.com)

Westminster, Colorado  
 PHONE: 303.379.3040

Houston, TX  
 713.784.3131

**Visual Numerics International Ltd.**

Sussex House  
 The Forbury  
 Reading, Berkshire RG1 3EJ  
 UNITED KINGDOM

PHONE: +44 118.925.3370  
 FAX: +44 118.925.3371  
 e-mail: [info@vniuk.co.uk](mailto:info@vniuk.co.uk)

**Visual Numerics SARL**

Immeuble le Wilson 1  
 33 Place des Corolles  
 F-92049 PARIS LA DEFENSE, Cedex  
 FRANCE

PHONE: +33-1-46-93-94-20  
 FAX: +33-1-46-93-94-39  
 e-mail: [info@vni.paris.fr](mailto:info@vni.paris.fr)

**Visual Numerics S. A. de C. V.**

Florencia 57 Piso 10-01  
 Col. Juarez  
 Mexico D. F. C. P. 06600  
 MEXICO

PHONE: +52-55-5514-9730 or 9628  
 FAX: +52-55-5514-5880  
[avadillo@mail.internet.com.mx](mailto:avadillo@mail.internet.com.mx)

**Visual Numerics International GmbH**

Zettachring 10  
 D-70567 Stuttgart  
 GERMANY

PHONE: +49-711-13287-0  
 FAX: +49-711-13287-99  
 e-mail: [info@visual-numerics.de](mailto:info@visual-numerics.de)

**Visual Numerics Japan, Inc.**

GOBANCHO HIKARI BLDG. 4<sup>TH</sup> Floor  
 14 GOBAN-CHO CHIYODA-KU  
 TOKYO, JAPAN 102-0076

PHONE: +81-3-5211-7760  
 FAX: +81-3-5211-7769  
 e-mail: [vnijapan@vnij.co.jp](mailto:vnijapan@vnij.co.jp)

**Visual Numerics, Inc.**

7/F, #510, Sect. 5  
 Chung Hsiao E. Road  
 Taipei, Taiwan 110  
 ROC

PHONE: (886) 22-727-2255  
 FAX: (886) 22-727-6798  
 e-mail: [info@vni.com.tw](mailto:info@vni.com.tw)

**Visual Numerics Korea, Inc.**

HANSHIN BLDG. Room 801  
 136-1, MAPO-DONG, MAPO-GU  
 SEOUL, 121-050  
 KOREA SOUTH

PHONE: +82-2-3273-2632 or 2633  
 FAX: +82-2-3273-2634  
 e-mail: [info@vni.co.kr](mailto:info@vni.co.kr)

World Wide Web site: <http://www.vni.com>

COPYRIGHT NOTICE: Copyright © 1990-2003, an unpublished work by Visual Numerics, Inc. All rights reserved. Printed in the USA.

The information contained in this document is subject to change without notice.

VISUAL NUMERICS, INC., MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual Numerics, Inc., shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance or use of this material.

IMSL, PV- WAVE, and Visual Numerics are registered in the U.S. Patent and Trademark Office by, and PV- WAVE Advantage is a trademark of, Visual Numerics, Inc.

TRADEMARK NOTICE: The following are trademarks or registered trademarks: Microsoft, Windows, Windows 95, Windows NT, Internet Explorer — Microsoft Corporation; Motif — The Open Systems Foundation, Inc.; PostScript — Adobe Systems, Inc.; UNIX — X/Open Company, Limited; X Window System, X11 — Massachusetts Institute of Technology; RISC System/6000 and IBM — International Business Machines Corporation; Sun, Java, JavaBeans — Sun Microsystems, Inc.; JavaScript, Netscape Communicator — Netscape, Inc.; HPGL and PCL — Hewlett Packard Corporation; DEC, VAX, VMS, OpenVMS — Digital Equipment Corporation; Tektronix 4510 Rasterizer — Tektronix, Inc.; IRIX, TIFF — Silicon Graphics, Inc.; SPARCstation — SPARC International, licensed exclusively to Sun Microsystems, Inc.; HyperHelp — Bristol Technology, Inc.

Use of this document is governed by a Visual Numerics Software License Agreement. This document contains confidential and proprietary information constituting valuable trade secrets. No part of this document may be reproduced or transmitted in any form without the prior written consent of Visual Numerics.

RESTRICTED RIGHTS NOTICE: This documentation is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the US Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFAR 252.227-7013, and in subparagraphs (a) through (d) of the Commercial Computer software — Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR Supplement, when applicable. Contractor/Manufacturer is Visual Numerics, Inc., 2000 Crow Canyon Place, Suite 270, San Ramon, CA 94583.

**IMSL** Fortran, C, and Java  
 Application Development Tools



# Contents

|                   |  |          |
|-------------------|--|----------|
| <b>Chapter 1:</b> | <b>Introduction</b> .....              | <b>1</b> |
|                   | Overview .....                         | 1        |
|                   | Implicitly Created Nodes .....         | 4        |
|                   | Adding a Chart to an Application ..... | 8        |
| <b>Chapter 2:</b> | <b>Charting Types</b> .....            | <b>9</b> |
|                   | Scatter Plot .....                     | 10       |
|                   | Simple Scatter Plot .....              | 10       |
|                   | Complex Scatter Plot .....             | 12       |
|                   | Line Plot .....                        | 14       |
|                   | Simple Line Plot .....                 | 14       |
|                   | Mixed Line and Marker Plot .....       | 16       |
|                   | Area Plot .....                        | 18       |
|                   | Simple Area Plot .....                 | 18       |
|                   | Painted Area Example .....             | 20       |
|                   | Attribute Reference .....              | 21       |
|                   | Function Plot .....                    | 22       |
|                   | Example .....                          | 22       |
|                   | Histogram Example .....                | 23       |
|                   | Spline Chart .....                     | 24       |
|                   | Log and SemiLog Plot .....             | 26       |
|                   | SemiLog Plot .....                     | 26       |
|                   | Log-Log Plot .....                     | 28       |
|                   | Error Bar Plot .....                   | 30       |
|                   | Vertical Error Bars .....              | 30       |
|                   | Horizontal Error Bars .....            | 32       |
|                   | Mixed Error Bars .....                 | 34       |
|                   | High-Low-Close Plot .....              | 36       |
|                   | Example .....                          | 37       |
|                   | Candlestick Chart .....                | 40       |
|                   | Example .....                          | 41       |

|  |           |
|--|-----------|
| Pie Chart .....                          | 44        |
| Example .....                            | 44        |
| Box Plot .....                           | 46        |
| Example .....                            | 46        |
| Bar Chart .....                          | 53        |
| Simple Bar Chart .....                   | 53        |
| Grouped Bar Chart .....                  | 55        |
| Stacked Grouped Bar Chart .....          | 57        |
| Legend .....                             | 59        |
| Attribute BarGap .....                   | 61        |
| Attribute BarWidth .....                 | 61        |
| Contour Chart .....                      | 62        |
| Example .....                            | 62        |
| Histogram .....                          | 64        |
| Example .....                            | 64        |
| Polar Plot .....                         | 67        |
| Example .....                            | 68        |
| <b>Chapter 3: Drawing Elements .....</b> | <b>71</b> |
| Line Attributes .....                    | 72        |
| Attribute LineColor .....                | 72        |
| Attribute LineWidth .....                | 72        |
| Attribute LineDashPattern .....          | 72        |
| Samples .....                            | 72        |
| Marker Attributes .....                  | 73        |
| Attribute MarkerType .....               | 73        |
| Attribute MarkerColor .....              | 74        |
| Attribute MarkerSize .....               | 74        |
| Attribute MarkerThickness .....          | 74        |
| Attribute MarkerDashPattern .....        | 74        |
| Samples .....                            | 74        |
| Fill Area Attributes .....               | 75        |
| FillOutlineType .....                    | 75        |
| FillOutlineColor .....                   | 75        |
| FillType .....                           | 75        |
| FillColor .....                          | 75        |
| Gradient .....                           | 75        |

|   |            |
|---|------------|
| FillPaint . . . . .                             | 78         |
| Text Attributes . . . . .                       | 79         |
| Attribute Font . . . . .                        | 79         |
| Attribute FontName . . . . .                    | 79         |
| Attribute FontSize . . . . .                    | 79         |
| Attribute FontStyle . . . . .                   | 79         |
| Samples . . . . .                               | 80         |
| Attribute TextAngle . . . . .                   | 80         |
| Attribute TextColor . . . . .                   | 80         |
| Attribute TextFormat . . . . .                  | 80         |
| Attribute Title . . . . .                       | 81         |
| Labels . . . . .                                | 82         |
| Attribute LabelType . . . . .                   | 82         |
| Data Point Labeling Example . . . . .           | 82         |
| Annotation . . . . .                            | 85         |
| AxisXY . . . . .                                | 87         |
| Axis Layout . . . . .                           | 88         |
| Transform . . . . .                             | 89         |
| Autoscale . . . . .                             | 89         |
| Background . . . . .                            | 119        |
| Solid Color Background . . . . .                | 119        |
| Gradient Color Background . . . . .             | 121        |
| Pattern Background . . . . .                    | 123        |
| Legend . . . . .                                | 125        |
| Simple Legend Example . . . . .                 | 125        |
| Legend Example . . . . .                        | 127        |
| Tool Tips . . . . .                             | 129        |
| Example . . . . .                               | 129        |
| <b>Chapter 4: XML . . . . .</b>                 | <b>131</b> |
| Chart XML Syntax . . . . .                      | 131        |
| Syntax Example . . . . .                        | 133        |
| Attribute Tags . . . . .                        | 134        |
| Array Tags . . . . .                            | 135        |
| Array Tag Examples . . . . .                    | 136        |
| Creating a Chart from XML . . . . .             | 136        |
| Interacting with an XML Created Chart . . . . . | 137        |

|   |            |
|---|------------|
| Enumerated Types in Chart XML Files .....                       | 137        |
| Marker Types .....  | 137        |
| Fill Types .....  | 138        |
| Text Alignments .....   | 138        |
| Font Styles .....   | 138        |
| Axis Parameters .....   | 138        |
| Autoscale Parameters .....                                      | 139        |
| Axis Transforms .....   | 139        |
| Data Node Types .....   | 139        |
| Label Types .....   | 139        |
| Bar Types .....   | 139        |
| Creating Charts from General XML Files Using XSLT .....         | 140        |
| Example .....   | 140        |
| XML Examples .....  | 146        |
| <b>Chapter 5: Actions .....</b>                                 | <b>169</b> |
| Picking .....   | 169        |
| Example .....   | 170        |
| Zoom .....  | 172        |
| Example .....   | 172        |
| Printing .....  | 178        |
| Printing from JFrameChart .....                                 | 178        |
| Printable Interface .....                                       | 178        |
| Pageable .....  | 178        |
| Serialization .....   | 180        |
| Example .....   | 180        |
| <b>Appendix A: Web Server Application .....</b>                 | <b>183</b> |
| Overview—Web Servlets .....                                     | 183        |
| Generating a Chart with a JSP .....                             | 184        |
| Generating Charts with Client-side Imagemaps .....              | 186        |
| Servlet Deployment .....  | 189        |
| <b>Appendix B: Writing a Chart as a Bitmap Image File .....</b> | <b>191</b> |
| Using the ImageIO Class .....                                   | 191        |
| Using the Java Advanced Imaging API .....                       | 192        |
| Using the Scalable Vector Graphics (SVG) API .....              | 194        |
| <b>Appendix C: Picture-in-Picture .....</b>                     | <b>197</b> |

**Index ..... I-1**





# Chapter 1: Introduction

The *JMSL Numerical Library Chart Programmer's Guide* is an overview and discussion of the JMSL charting package. It is intended to be used in conjunction with the online *Javadoc*-generated Reference Manual (see <http://www.vni.com/products/jmsl/api/com/ims1/chart/package-summary.html>).

This guide is both an introductory tutorial on the charting package and a “cook-book” for creating common chart types. The Reference Manual contains the complete details of the classes.

---

**NOTE:** Charts in the printed version of this manual appear in black and white. If you are reading this manual in print and wish to see the charts in color, please open the Acrobat Reader file:

[/manual/chartpg](#)

If you do not have Acrobat Reader installed on your machine, you can download it for free at:

<http://www.adobe.com/products/acrobat/readstep2.html>

Or you may also view the *Guide* in color from our Web site at:

<http://www.vni.com/products/jmsl/chartpg/index.html>

---

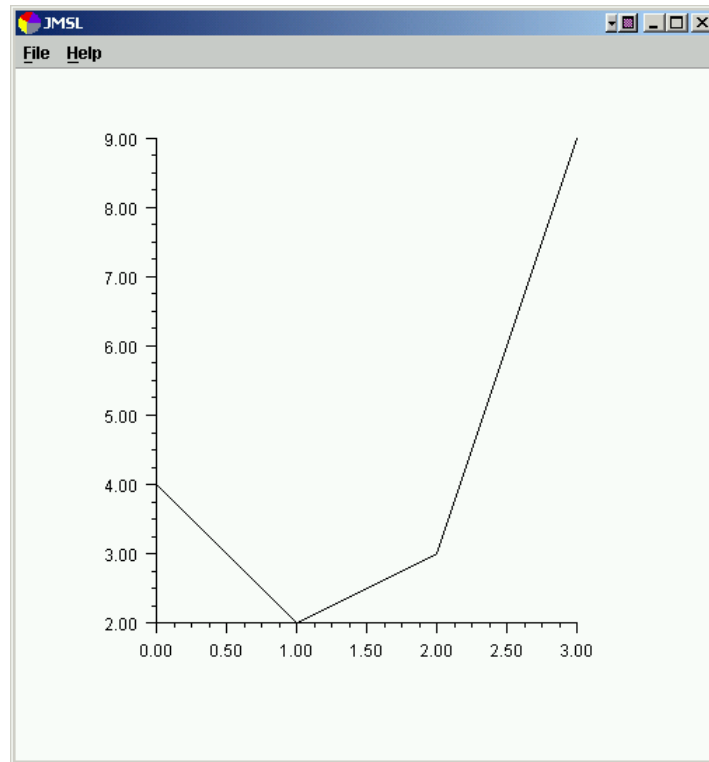
---

## Overview

The JMSL chart package is designed to allow the creation of highly customizable charts using Java. A JMSL chart is created by assembling ChartNodes into a tree. This chart tree is then rendered to the screen or printer.

The following class is a simple example of the use of the JMSL chart package. The chart is displayed in a Swing frame. The code to create the chart is all in the con-

structor. The JMSL class `JFrameChart` extends the Swing class `JFrame` and creates a `Chart` object.



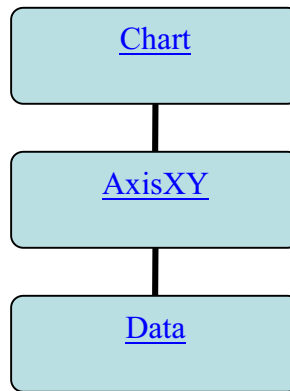
```
import com.imsl.chart.*;

public class Introl extends JFrameChart {

    public Introl() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);
    }

    public static void main(String argv[]) {
        new Introl().setVisible(true);
    }
}
```

The above example created and assembled three nodes into the following tree. The root Chart node is created by the JFrameChart class.



The general pattern of the ChartNode class, and classes derived from it, is to have constructors whose first argument is the parent ChartNode of the ChartNode being created.

The root node of the tree is always a Chart object. It is usually constructed within JFrameChart or JPanelChart, which handle the repainting of the chart within Swing. If a Chart object is explicitly created, and used within a Swing GUI, then its `paint(Graphics)` method must be called from the container's `paint(Graphics)` or `paintComponent(Graphics)` methods.

Chart nodes can contain attributes, such as `FillColor` and `LineWidth`. Attributes are inherited via the chart tree. For example, when a Data node is being painted, its `LineWidth` attribute determines the thickness of the line. If this attribute is set in the data node being drawn, that is the value used. If it is not set, then its parent node (an AxisXY node in the above example) is queried. If it is not set there, then *its* parent is queried. This continues until the root node is reached after which a default value is used. Note that this inheritance is not the same as Java class inheritance.

Attributes are set using setter methods, such as `setLineWidth(double)` and retrieved using getter methods, such as `getLineWidth(double)`.

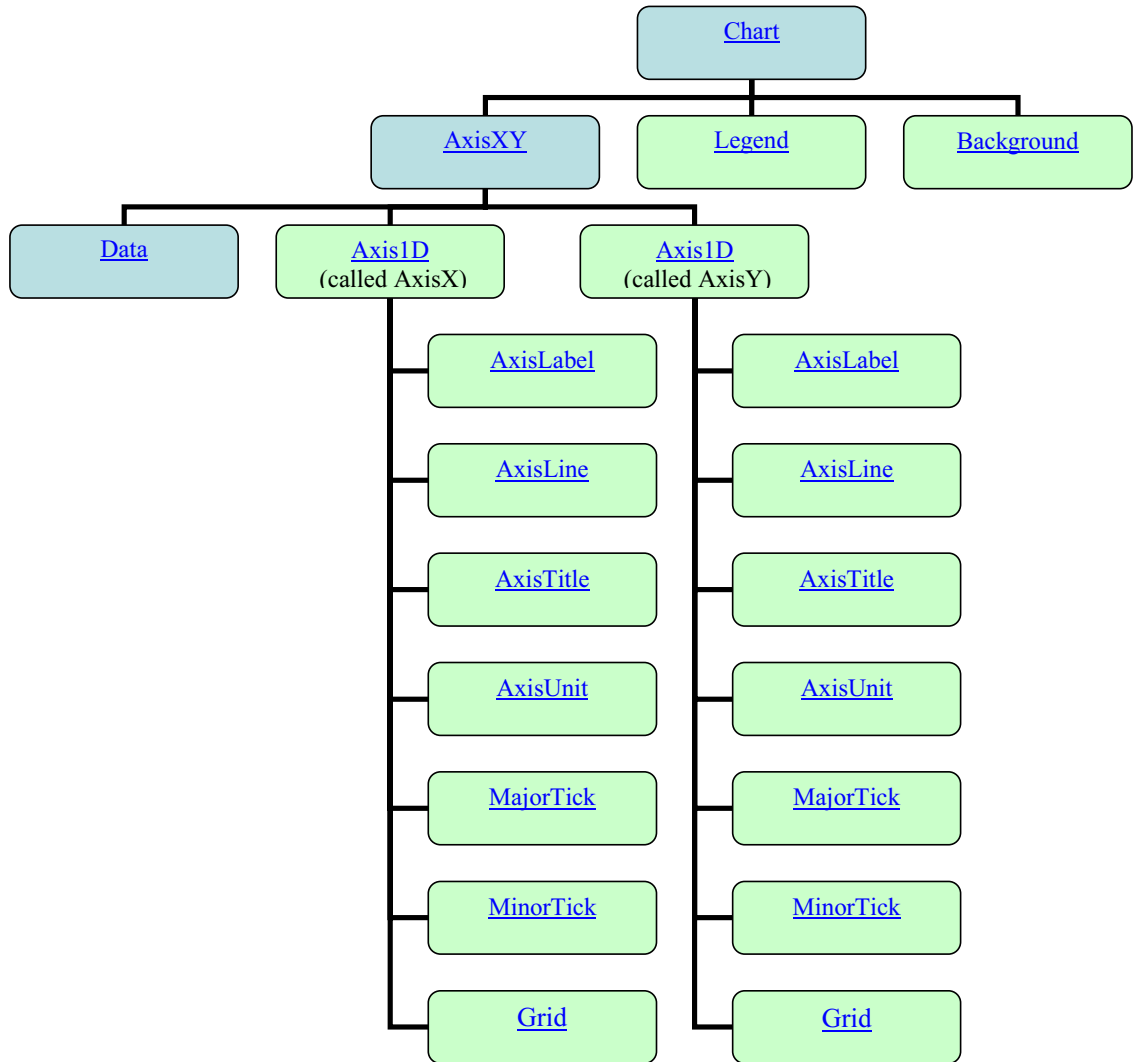
## Implicitly Created Nodes

Some chart nodes automatically create additional nodes, as their children, when they are created. These nodes can be accessed via `get` methods. For example, the code

```
chart.setBackground().setFillColor(java.awt.Color.green);
```

changes the background color. The method `setBackground` retrieves the `Background` node from the chart object and the method `setFillColor` sets the `Background` object's `FillColor` attribute to `Color.green`.

In the following diagram, the nodes automatically created are shown in light green (to view in color see the online documentation).

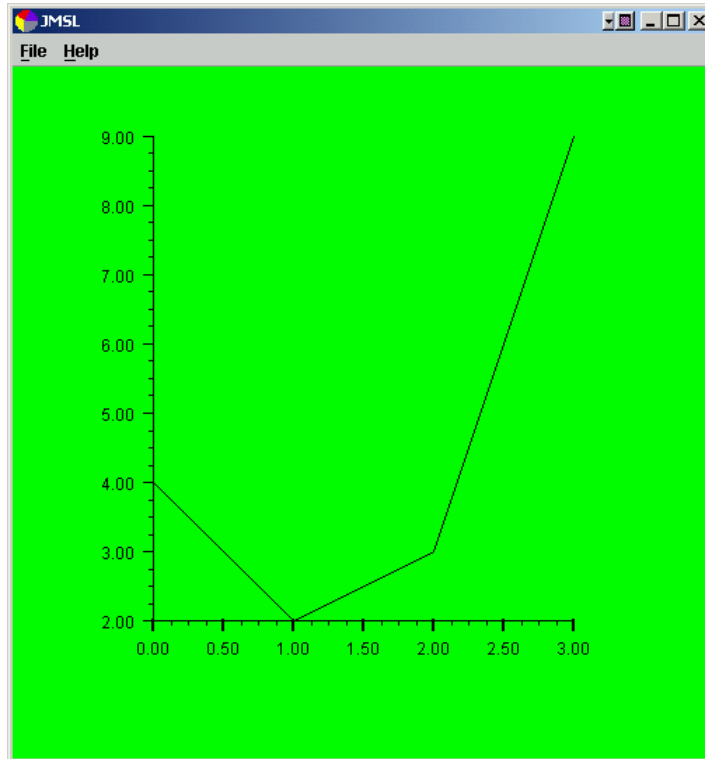


Method calls can be chained together. For example, the following sets the thickness of the major tick marks on the y-axis.

```
axis.getAxisX().getMajorTick().setLineWidth(2.0);
```

where `axis` is an `AxisXY` object.

A customized version of the above chart can be obtained by changing its constructor as in the following.



```
import com.ims1.chart.*;

public class Intro2 extends JFrameChart {

    public Intro2() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);

        chart.getBackground().setFillColor(java.awt.Color.green);
        axis.getAxisX().getMajorTick().setLineWidth(2.0);
    }

    public static void main(String argv[]) {
        new Intro2().setVisible(true);
    }
}
```



## Adding a Chart to an Application

For simplicity, most of the examples in this manual use the `JFrameChart` class. `JFrameChart` is useful for quickly building an application that is a chart. The class `JPanelChart` is used to build a chart into a larger application. It extends Swing's `JPanel` class and can be used wherever a `JPanel` can be used. The following code shows a `JPanelChart` being created, added to a `JFrame`, and having a chart tree added to the `JPanelChart`. (The generated chart is very similar to that shown at the beginning of this chapter.)

```
import com.imsl.chart.*;

public class SampleJPanel extends javax.swing.JFrame {
    private JPanelChart jPanelChart;

    public SampleJPanel() {
        this.setSize(500, 500);
        jPanelChart = new JPanelChart();
        getContentPane().add(jPanelChart);

        Chart chart = jPanelChart.getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);
    }

    public static void main(String argv[]) {
        new SampleJPanel().setVisible(true);
    }
}
```

# Chapter 2: Charting Types

This chapter describes these JMSL charting types:

- [“Scatter Plot” on page 10](#)
- [“Line Plot” on page 14](#)
- [“Function Plot” on page 22](#)
- [“Log and SemiLog Plot” on page 26](#)
- [“Error Bar Plot” on page 30](#)
- [“High-Low-Close Plot” on page 36](#)
- [“Candlestick Chart” on page 40](#)
- [“Pie Chart” on page 44](#)
- [“Box Plot” on page 46](#)
- [“Bar Chart” on page 53](#)
- [“Contour Chart” on page 62](#)
- [“Histogram” on page 64](#)
- [“Polar Plot” on page 67](#)

---

## Scatter Plot

This section describes the construction of scatter charts. The markers can be formatted using the [Marker Attributes](#) (see [page 73](#)).

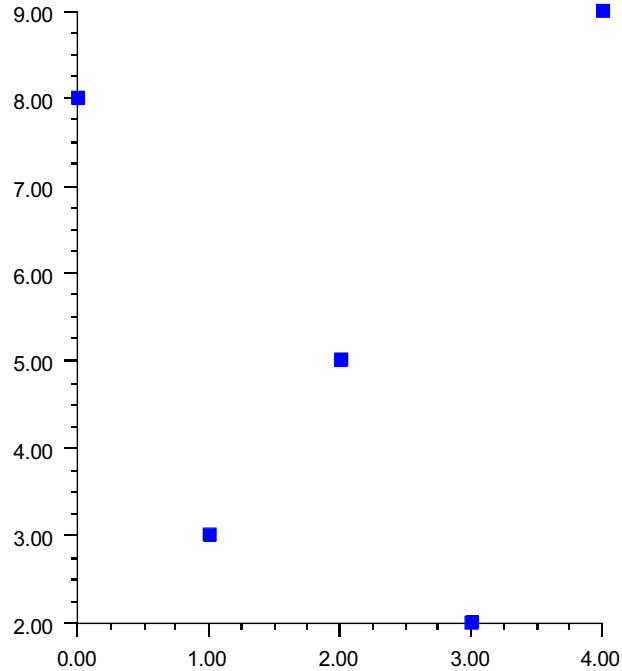
It is also possible to mix lines and markers (see [“Mixed Line and Marker Plot” on page 16](#)).

### Simple Scatter Plot

The `JFrameChart` class is used to create a frame containing a `Chart` node. The `Chart` node is the root of the tree to which an `AxisXY` node is added. A `Data` node is then created as the child of the axis node. The `Data` node is created using an array of  $y$ -values. The  $x$ -values default to 0, 1, ....

The `DataType` attribute is set to `DATA_TYPE_MARKER` to make this a scatter plot.

The look of the markers is controlled by the marker attributes. In this example the `MarkerType` attribute is set to `MARKER_TYPE_FILLED_SQUARE`. The `MarkerColor` attribute is set to blue (to view plot in color see [online documentation](#)).



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleSimpleScatter extends JFrameChart {

    public SampleSimpleScatter() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double y[] = {8, 3, 5, 2, 9};
        Data datal = new Data(axis, y);
        datal.setDataType(Data.DATA_TYPE_MARKER);
        datal.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
        datal.setMarkerColor(Color.blue);
    }

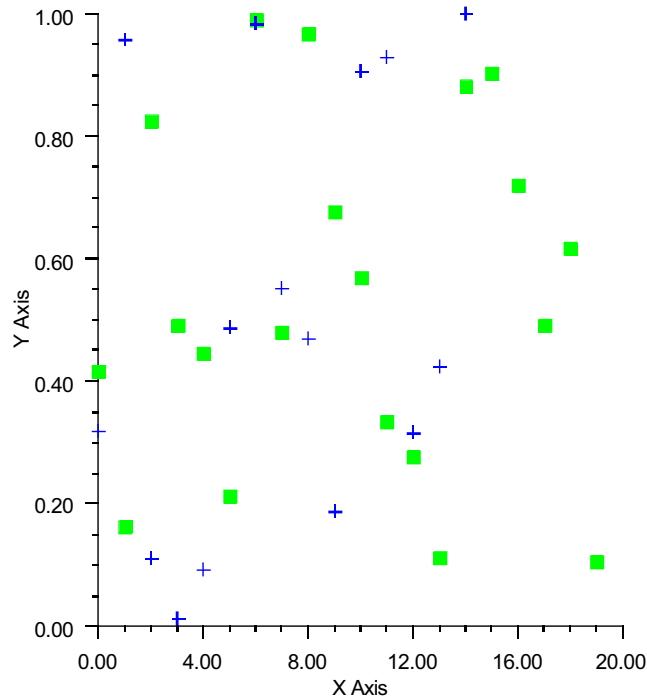
    public static void main(String argv[]) {
        new SampleSimpleScatter().setVisible(true);
    }
}
```

## Complex Scatter Plot

This example shows a scatter plot with two data sets. The Data nodes are created using random  $y$ -values generated by Random. The  $x$ -values default to 0, 1, ....

The axes are labeled by setting the AxisTitle attribute for the  $x$  and  $y$  axes.

The DataType attribute is set in the axis node. The axis node does not itself use this attribute, but from there it is inherited by the child Data nodes.



```

import com.imsl.chart.*;
import java.awt.Color;
import com.imsl.stat.Random;

public class SampleScatter extends JFrameChart {

    public SampleScatter() {
        Random r = new Random(123457);

        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        axis.getAxisX().getAxisTitle().setTitle("X Axis");
        axis.getAxisY().getAxisTitle().setTitle("Y Axis");
        axis.setDataTypes(Data.DATA_TYPE_MARKER);

        // Data set 1
        double y1[] = new double[20];
        for (int k = 0; k < y1.length; k++) y1[k] = r.nextDouble();
        Data data1 = new Data(axis, y1);
        data1.setDataTypes(Data.DATA_TYPE_MARKER);
        data1.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
        data1.setMarkerColor(Color.green);

        // Data set 2
        double y2[] = new double[15];
        for (int k = 0; k < y2.length; k++) y2[k] = r.nextDouble();
        Data data2 = new Data(axis, y2);
        data2.setMarkerType(Data.MARKER_TYPE_PLUS);
        data2.setMarkerColor(Color.blue);
    }

    public static void main(String argv[]) {
        new SampleScatter().setVisible(true);
    }
}

```

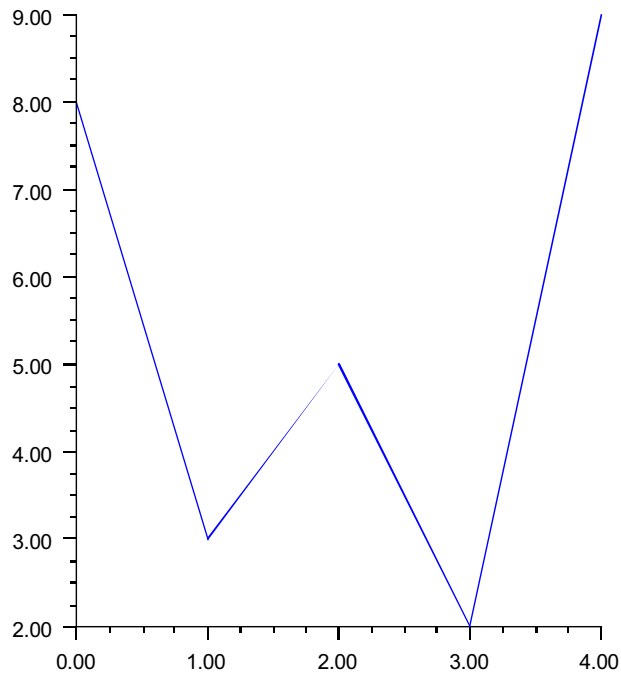
---

## Line Plot

A line plot consists of points connected by lines. The lines can be formatted using the [Line Attributes](#) (see [page 72](#)).

### Simple Line Plot

This example shows a simple line plot. The Data node is created using an array of  $y$ -values. The  $x$ -values default to 0, 1, .... The `DataType` attribute is set to `DATA_TYPE_LINE` to make this a line chart. The look of the line is controlled by the line attributes. Here the `LineColor` attribute is set to blue.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleSimpleLine extends JFrameChart {

    public SampleSimpleLine() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

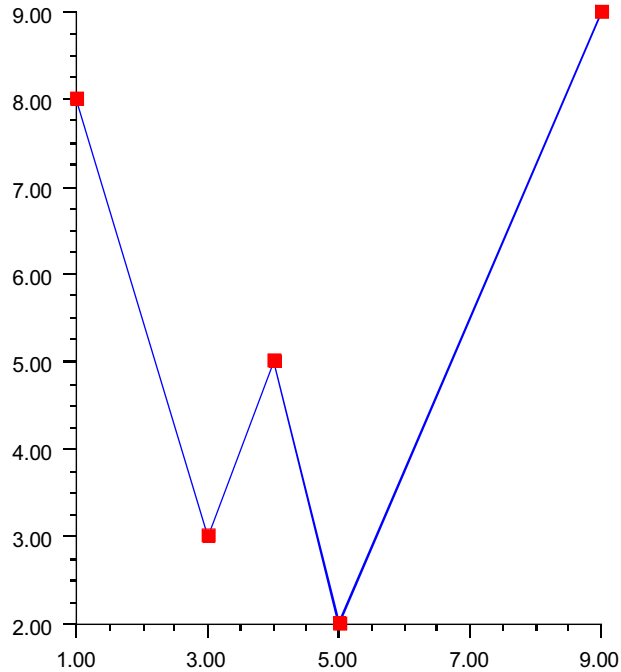
        double y[] = {8, 3, 5, 2, 9};
        Data data1 = new Data(axis, y);
        data1.setDataType(Data.DATA_TYPE_LINE);
        data1.setLineColor(Color.blue);
    }

    public static void main(String argv[]) {
        new SampleSimpleLine().setVisible(true);
    }
}
```



## Mixed Line and Marker Plot

The `DataType` attribute can be set using “or” syntax to combine types. In this example, it is set to `DATA_TYPE_LINE | DATA_TYPE_MARKER`. This example also explicitly sets both the `x` and the `y` values of the data points. Note that the `x` values do not have to be uniformly spaced.



```

import com.imsl.chart.*;

public class SampleLineMarker extends JFrameChart {

    public SampleLineMarker() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double x[] = {1, 3, 4, 5, 9};
        double y[] = {8, 3, 5, 2, 9};
        Data data = new Data(axis, x, y);
        data.setDataType(Data.DATA_TYPE_LINE | Data.DATA_TYPE_MARKER);
        data.setLineColor("blue");
        data.setMarkerColor("red");
        data.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
    }

    public static void main(String argv[]) {
        new SampleLineMarker().setVisible(true);
    }
}

```

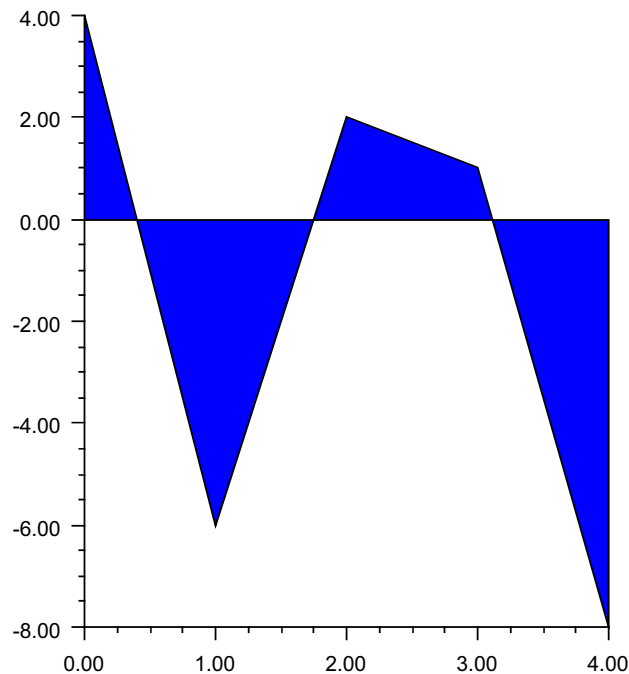
---

## Area Plot

Area plots are similar to line plots, but with the area between the line and a reference line filled in. An area plot is created if the `DATA_TYPE_FILL` bit is on in the value of the `DataType` attribute. The default reference line is  $y=0$ . The location of the reference line can be changed from 0 by using `setReference(double)`. The [Fill Area Attributes](#) determine how the area is filled (see [page 75](#)).

### Simple Area Plot

This example draws a simple area plot. The default value of the `FillType` attribute is `FILL_TYPE_SOLID`. The example sets the `FillColor` attribute to blue. So the area between the line and  $y=0$  is solid blue.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleArea extends JFrameChart {

    public SampleArea() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double y[] = {4, -6, 2, 1, -8};
        Data data = new Data(axis, y);
        data.setDataType(Data.DATA_TYPE_FILL);
        data.setFillColor(Color.blue);
    }

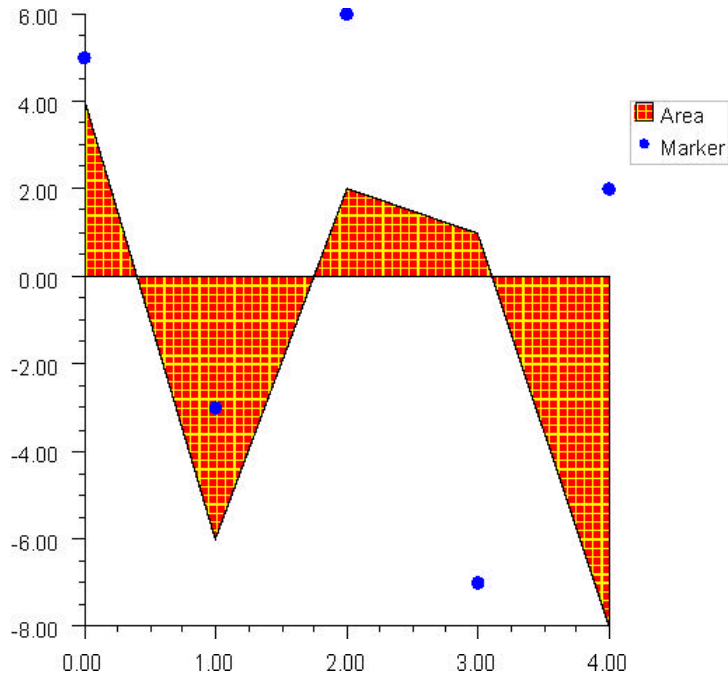
    public static void main(String argv[]) {
        new SampleArea().setVisible(true);
    }
}
```

## Painted Area Example

This example shows an area chart filled in with a painted texture. The texture is created by a static method `FillPaint.crosshatch`.

A second data set is plotted as a set of markers.

The Legend node is painted in this example and has entries for both the filled area data set and the marker data set.



```

import com.imsl.chart.*;
import java.awt.Color;

public class SampleAreaPaint extends JFrameChart {

    public SampleAreaPaint() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        chart.getLegend().setPaint(true);

        double y1[] = {4, -6, 2, 1, -8};
        Data data1 = new Data(axis, y1);
        data1.setTitle("Area");
        data1.setDataType(Data.DATA_TYPE_FILL);
        data1.setFillType(Data.FILL_TYPE_PAINT);
        data1.setFillPaint(FillPaint.crosshatch
            (10,5,Color.red,Color.yellow));

        double y2[] = {5, -3, 6, -7, 2};
        Data data2 = new Data(axis, y2);
        data2.setTitle("Marker");
        data2.setDataType(Data.DATA_TYPE_MARKER);
        data2.setMarkerColor(Color.blue);
        data2.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
    }

    public static void main(String argv[]) {
        new SampleAreaPaint().setVisible(true);
    }
}

```

## Attribute Reference

The attribute Reference defines the reference line. If its value is  $a$ , then the reference line is  $y = a$ . Its default value is 0.

---

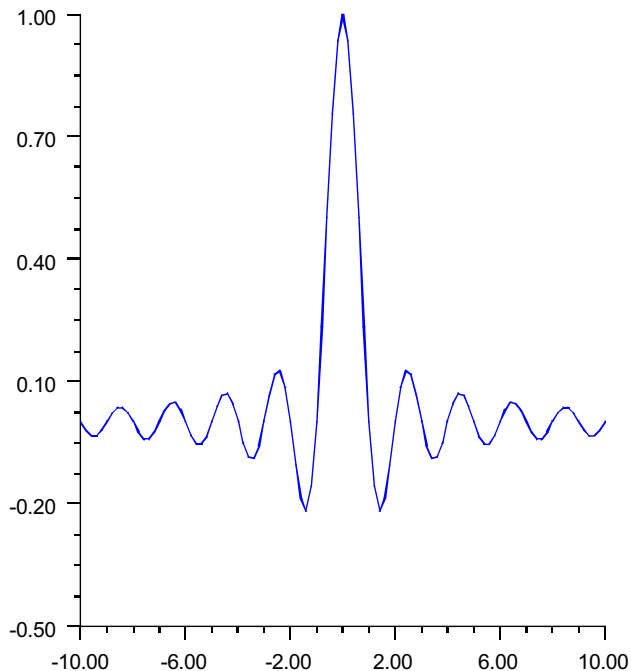
## Function Plot

A function plot shows the value of a function  $f(x)$  over an interval  $[a,b]$ . The function must be defined as an implementation of the `ChartFunction` interface. A `Data` node constructor creates a line chart from the function. The look of the function is controlled by the line attributes.

The `ChartFunction` interface requires that the function name be "f", that the function has a single double argument and that it returns a double.

### Example

This example plots the sinc function on the interval  $[-10,10]$ . The sinc function is defined to be  $\sin(\pi x)/\pi x$ . In this example, `sinc` is an anonymous inner class that implements `ChartFunction`. This is required by the function `Data` constructor. In the code, the case  $x=0$  is handled specially to avoid returning NaN.



```

import com.imsl.chart.*;
import java.awt.Color;

public class SampleFunction extends JFrameChart {

    public SampleFunction() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        ChartFunction sinc = new ChartFunction() {
            public double f(double x) {
                if (x == 0.0) return 1.0;
                return Math.sin(Math.PI*x) / (Math.PI*x);
            }
        };
        Data data = new Data(axis, sinc, -10.0, 10.0);
        data.setLineColor(Color.blue);
    }

    public static void main(String argv[]) {
        new SampleFunction().setVisible(true);
    }
}

```

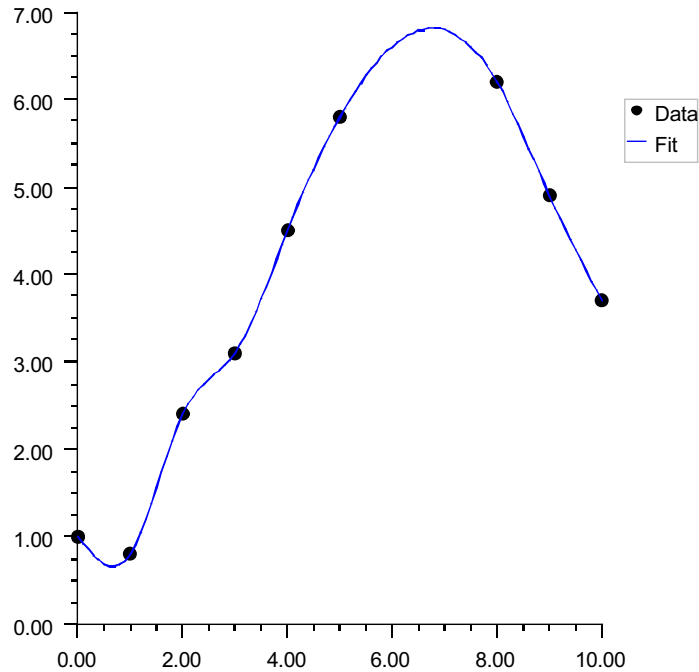
## Histogram Example

For another example of a ChartFunction, see [“Histogram” on page 64](#).



## Spline Chart

This example shows raw data points, as markers, and their fit to a shape preserving spline. The spline is computed using `CsShape` found in the JMSL Math package (which extends `Spline`). The `ChartSpline` class wraps the `Spline` into a `ChartFunction`. This example also enables the Legend.



```

import com.imsl.chart.*;
import com.imsl.math.CsShape;
import java.awt.Color;

public class SampleSpline extends JFrameChart {

    public SampleSpline() {
        try {
            Chart chart = getChart();
            AxisXY axis = new AxisXY(chart);
            chart.getLegend().setPaint(true);

            double x[] = {0, 1, 2, 3, 4, 5, 8, 9, 10};
            double y[] = {1.0, 0.8, 2.4, 3.1, 4.5, 5.8, 6.2, 4.9, 3.7};
            Data dataMarker = new Data(axis, x, y);
            dataMarker.setTitle("Data");
            dataMarker.setDataType(Data.DATA_TYPE_MARKER);
            dataMarker.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);

            CsShape spline = new CsShape(x, y);
            Data dataSpline = new Data(axis, new ChartSpline(spline), 0., 10.);
            dataSpline.setTitle("Fit");
            dataSpline.setLineColor(Color.blue);
        } catch (com.imsl.IMSLException e) {
            e.printStackTrace();
        }
    }

    public static void main(String argv[]) {
        new SampleSpline().setVisible(true);
    }
}

```

---

## Log and SemiLog Plot

In a semilog plot the y-axis is logarithmically scaled, while the x-axis is linearly scaled. In a log-log plot both axes are logarithmically scaled.

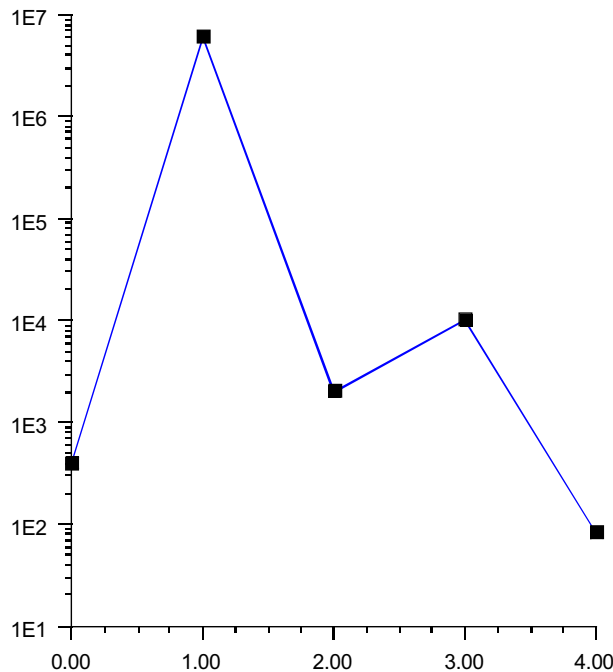
### SemiLog Plot

In this example data is plotted as lines and markers on a semilog axis.

To set up the y-axis as a logarithmic axis,

- The Transform attribute is set to `TRANSFORM_LOG`.
- The Density attribute is set to 9. Density is the number of minor ticks between major ticks. It is 9 because base 10 is used and the major tick marks are not counted.

The TextFormat, used by AxisLabel, is changed to use scientific notation (see [“Text Attributes” on page 79](#)). The default decimal format would result in large numbers written with many zeros.



```

import com.imsl.chart.*;
import java.awt.Color;

public class SampleSemiLog extends JFrameChart {

    public SampleSemiLog() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        axis.getAxisY().setTransform(axis.TRANSFORM_LOG);
        axis.getAxisY().setDensity(9);
        axis.getAxisY().setTextFormat(new java.text.DecimalFormat("0.E0"));

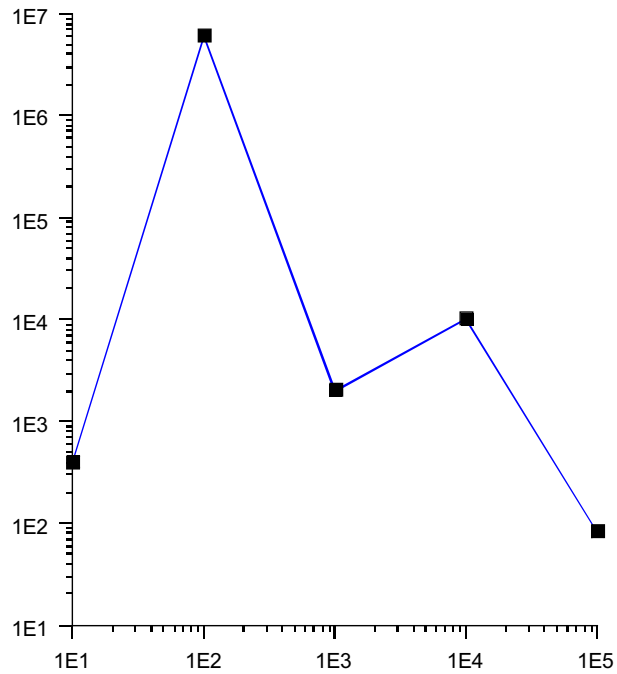
        double y[] = {4e2, 6e6, 2e3, 1e4, 8e1};
        Data data = new Data(axis, y);
        data.setDataType(Data.DATA_TYPE_LINE | Data.DATA_TYPE_MARKER);
        data.setLineColor(Color.blue);
        data.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
    }

    public static void main(String argv[]) {
        new SampleSemiLog().setVisible(true);
    }
}

```

## Log-Log Plot

A log-log plot is set up in much the same way as a semilog plot, except that the changes must be made to both the  $x$ -axis and the  $y$ -axis. In this example, the changes are made to the axis node and inherited by both the  $x$  and  $y$ -axis nodes.



```

import com.imsl.chart.*;
import java.awt.Color;

public class SampleLogLog extends JFrameChart {

    public SampleLogLog() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        axis.setTransform(axis.TRANSFORM_LOG);
        axis.setDensity(9);
        axis.setTextFormat(new
java.text.DecimalFormat("0.E0"));

        double x[] = {1e1, 1e2, 1e3, 1e4, 1e5};
        double y[] = {4e2, 6e6, 2e3, 1e4, 8e1};
        Data data = new Data(axis, x, y);
        data.setDataType(Data.DATA_TYPE_LINE |
Data.DATA_TYPE_MARKER);
        data.setLineColor(Color.blue);
        data.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
    }

    public static void main(String argv[]) {
        new SampleLogLog().setVisible(true);
    }
}

```

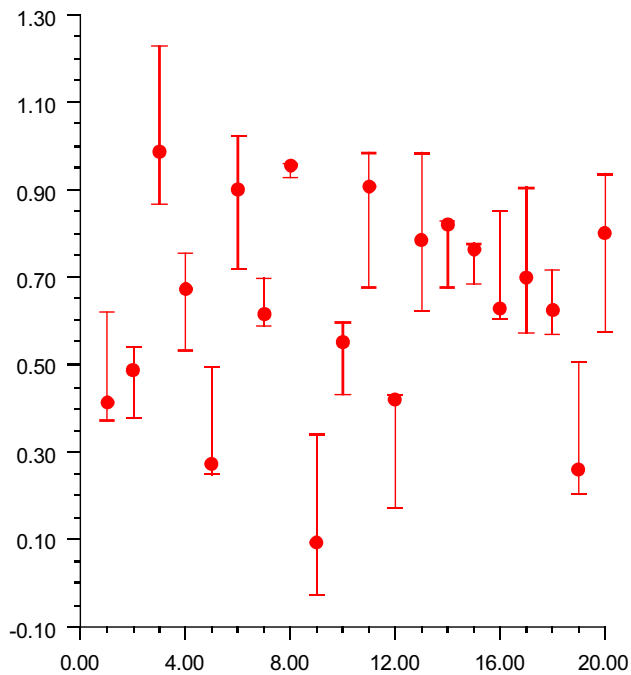
---

## Error Bar Plot

Error bars are used to indicate the estimated error in a measurement. Errors bars indicate the uncertainty in the  $x$  and/or  $y$  values.

### Vertical Error Bars

The most common error bar plot is one in which the errors are in the  $y$ -values. This example shows such an error bar plot. Note that the values of the low and high arguments are absolute  $y$  values, not relative or percentage values.



```

import com.imsl.chart.*;
import java.awt.Color;
import com.imsl.stat.Random;

public class SampleErrorBar extends JFrameChart {

    public SampleErrorBar() {
        Random r = new Random(123457);

        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        // Generate a random data set, with random errors
        int n = 20;
        double x[] = new double[n];
        double y[] = new double[n];
        double low[] = new double[n];
        double high[] = new double[n];
        for (int k = 0; k < n; k++) {
            x[k] = k + 1;
            y[k] = r.nextDouble();
            low[k] = y[k] - 0.25*r.nextDouble();
            high[k] = y[k] + 0.25*r.nextDouble();
        }

        ErrorBar data = new ErrorBar(axis, x, y, low, high);
        data.setDataType(data.DATA_TYPE_ERROR_Y | data.DATA_TYPE_MARKER);
        data.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
        data.setMarkerColor(Color.red);
    }

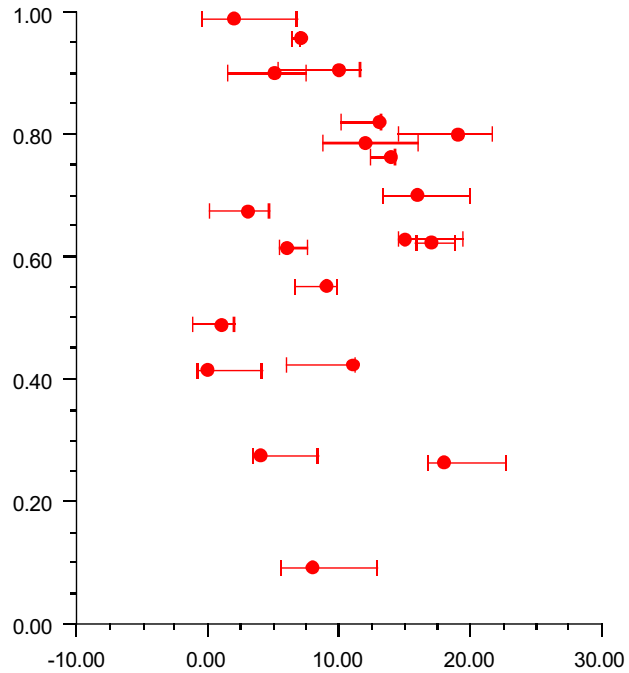
    public static void main(String argv[]) {
        new SampleErrorBar().setVisible(true);
    }
}

```



## Horizontal Error Bars

It is also possible to have horizontal error bars, indicating errors in  $x$ , as shown in this example.



```

import com.imsl.chart.*;
import java.awt.Color;
import com.imsl.stat.Random;

public class SampleHorizontalErrorBar extends JFrameChart {

    public SampleHorizontalErrorBar() {
        Random r = new Random(123457);

        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        // Generate a random data set, with random errors in x
        int n = 20;
        double x[] = new double[n];
        double y[] = new double[n];
        double low[] = new double[n];
        double high[] = new double[n];
        for (int k = 0; k < n; k++) {
            x[k] = k;
            y[k] = r.nextDouble();
            low[k] = x[k] - 5.*r.nextDouble();
            high[k] = x[k] + 5.*r.nextDouble();
        }

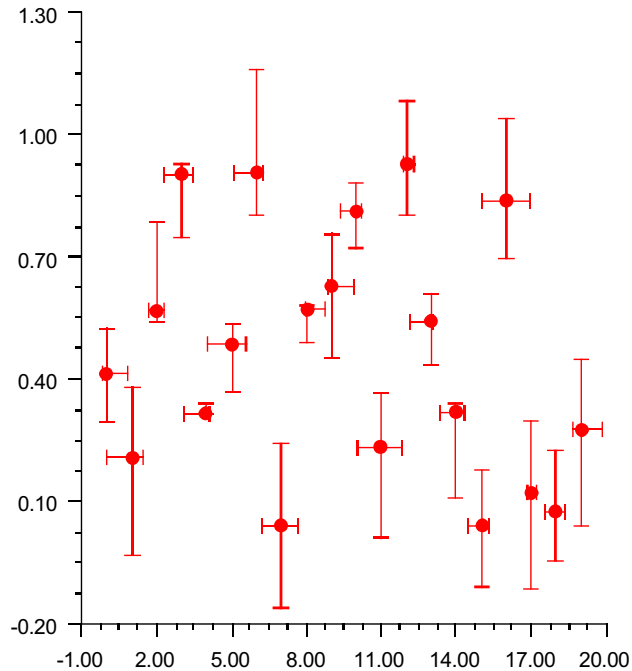
        ErrorBar data = new ErrorBar(axis, x, y, low, high);
        data.setDataType(data.DATA_TYPE_ERROR_X | data.DATA_TYPE_MARKER);
        data.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
        data.setMarkerColor("red");
    }

    public static void main(String argv[]) {
        new SampleHorizontalErrorBar().setVisible(true);
    }
}

```

## Mixed Error Bars

To show errors in both  $x$  and  $y$ , it is necessary to create both vertical and horizontal error bar objects. This example shows such a chart.



```
import com.imsl.chart.*;
import java.awt.Color;
import com.imsl.stat.Random;

public class SampleMixedErrorBar extends JFrameChart {

    public SampleMixedErrorBar() {
        Random r = new Random(123457);

        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        // Generate a random data set, with random errors in x
        int n = 20;
        double x[] = new double[n];
        double y[] = new double[n];
        double xlow[] = new double[n];
        double xhigh[] = new double[n];
        double ylow[] = new double[n];
```

```

double yhigh[] = new double[n];
for (int k = 0; k < n; k++) {
    x[k] = k;
    y[k] = r.nextDouble();
    xlow[k] = x[k] - r.nextDouble();
    xhigh[k] = x[k] + r.nextDouble();
    ylow[k] = y[k] - 0.25*r.nextDouble();
    yhigh[k] = y[k] + 0.25*r.nextDouble();
}

ErrorBar dataY = new ErrorBar(axis, x, y, ylow, yhigh);
dataY.setDataType(dataY.DATA_TYPE_ERROR_Y | dataY.DATA_TYPE_MARKER);
dataY.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
dataY.setMarkerColor("red");

ErrorBar dataX = new ErrorBar(axis, x, y, xlow, xhigh);
dataX.setDataType(dataY.DATA_TYPE_ERROR_X);
dataX.setMarkerColor("red");
}

public static void main(String argv[]) {
    new SampleMixedErrorBar().setVisible(true);
}
}

```

---

## High-Low-Close Plot

High-Low-Close plots are used to show stock prices. They are created using the `HighLowClose` class.

The markers can be formatted using the attribute `MarkerColor`.

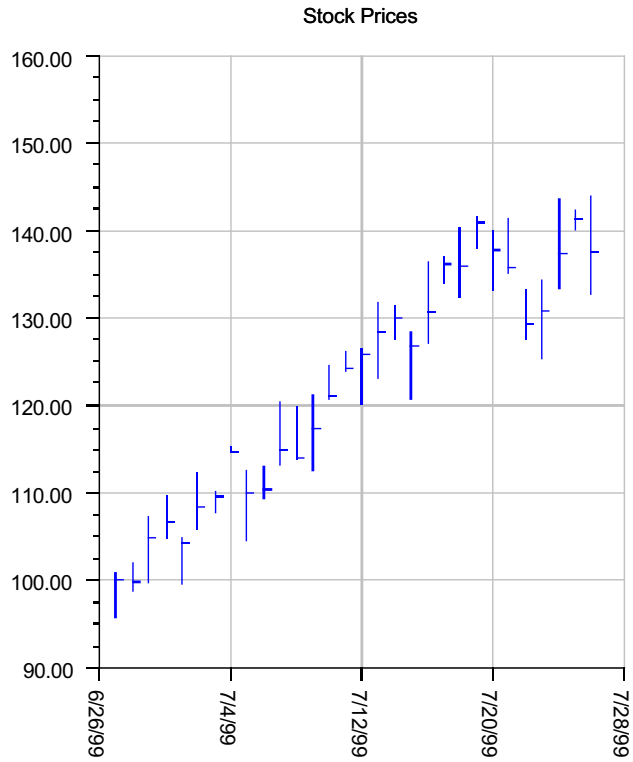
In a high-low-close plot the vertical line represents the high and low values. The close value is represented by a “tick” to the right. The open value, if present, is represented by a tick to the left.

The `setupDateAxis` method will configure the  $x$ -axis for dates. This turns off autoscaling of the axis.

In a high-low-close chart the  $x$ -values are the number of milliseconds since the epoch (January 1, 1970). This is the standard Java convention for representing dates as numbers.

## Example

In this example, random security prices are computed in the `createData` method. The time axis is prepared by calling `setDateAxis`.



```

import com.imsl.chart.*;
import java.text.DateFormat;
import java.util.*;

public class SampleHighLowClose extends JFrameChart {
    private double high[], low[], close[];

    public SampleHighLowClose() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        // Date is June 27, 1999
        Date date = new GregorianCalendar(1999,
        GregorianCalendar.JUNE, 27).getTime();

        int n = 30;
        createData(n);

        //Create an instance of a HighLowClose Chart
        HighLowClose hilo = new HighLowClose(axis, date, high,
low, close);
        hilo.setMarkerColor("blue");

        //Set the HighLowClose Chart Title
        chart.getChartTitle().setTitle("Stock Prices");

        // Setup the x axis
        Axis1D axisX = axis.getAxisX();

        //Set the text angle for the X axis labels
        axisX.getAxisLabel().setTextAngle(270);
        //Change the text format to the local date format
        DateFormat df =
DateFormat.getDateInstance(DateFormat.SHORT);
        axisX.getAxisLabel().setTextFormat(df);

        // Setup the time axis
        hilo.setDateAxis("Date(SHORT)");

        // Turn on grid and make it light gray
        axisX.getGrid().setPaint(true);
        axisX.getGrid().setLineColor("lightgray");
        axis.getAxisY().getGrid().setPaint(true);
        axis.getAxisY().getGrid().setLineColor("lightgray");
    }
}

```

```

private void createData(int n) {
    high = new double[n];
    low = new double[n];
    close = new double[n];
    Random r = new Random(123457);
    for (int k = 0; k < n; k++) {
        double f = r.nextDouble();
        if (k == 0) {
            close[0] = 100;
        } else {
            close[k] = (0.95+0.10*f)*close[k-1];
        }
        high[k] = close[k]*(1.+0.05*r.nextDouble());
        low[k] = close[k]*(1.-0.05*r.nextDouble());
    }
}

public static void main(String argv[]) {
    new SampleHighLowClose().setVisible(true);
}
}

```



---

## Candlestick Chart

Candlesticks are used to show stock price. Each candlestick shows the stock's high, low, opening and closing prices.

The Candlestick constructors create two child CandlestickItem nodes. One is for the up days and one is for the down days. A day is an up day if the closing price is greater than the opening price. The `getUp` and `getDown` accessor methods can be used to retrieve these nodes.

The line (“whisker”) part of a candlestick shows the stock's high and low prices. The whiskers can be formatted using the line attributes (see [“Line Attributes” on page 72](#)).

The body of a candlestick shows the stock's opening and closing prices. The body color is used to flag if the top of the body is the closing price (up day) or the opening price (down day). The fill area attributes determine how the body is drawn (see [“Fill Area Attributes” on page 75](#)). By default, up days are white and down days are black.

The width of a candlestick is controlled by the MarkerSize attribute (see [“Attribute MarkerSize” on page 74](#)).

## Example

In this example, random security prices are computed in the `createData` method. The time axis is prepared by calling `setDateAxis`.

The up days are colored green and the down days are colored red.



```
import com.imsl.chart.*;
import java.text.DateFormat;
import java.util.*;

public class SampleCandlestick extends JFrameChart {
    private double high[], low[], open[], close[];

    public SampleCandlestick() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        // Date is June 27, 1999
        Date date = new GregorianCalendar(1999, GregorianCal-
```

```

endar.JUNE, 27).getTime();

    int n = 30;
    createData(n);

    //Create an instance of a HighLowClose Chart
    Candlestick stick = new Candlestick(axis, date, high,
low, close, open);

    // show up days in green and down days in red
    stick.getUp().setFillColor("green");
    stick.getDown().setFillColor("red");

    //Set the HighLowClose Chart Title
    chart.getChartTitle().setTitle("Stock Prices");

    // Setup the x axis
    Axis1D axisX = axis.getAxisX();

    // Setup the time axis
    stick.setDateAxis("Date(SHORT)");

    // Turn on grid and make it light gray
    axisX.getGrid().setPaint(true);
    axisX.getGrid().setLineColor("lightgray");
    axis.getAxisY().getGrid().setPaint(true);
    axis.getAxisY().getGrid().setLineColor("lightgray");
}

private void createData(int n) {
    high = new double[n];
    low = new double[n];
    close = new double[n];
    open = new double[n];
    Random r = new Random(123457);
    for (int k = 0; k < n; k++) {
        double f = r.nextDouble();
        if (k == 0) {
            close[0] = 100;
        } else {
            close[k] = (0.95+0.10*f)*close[k-1];
        }
        high[k] = close[k]*(1.+0.05*r.nextDouble());
        low[k] = close[k]*(1.-0.05*r.nextDouble());
    }
}

```

```
        open[k] = low[k] + r.nextDouble()*(high[k]-low[k]);
    }
}

public static void main(String argv[]) {
    new SampleCandlestick().setVisible(true);
}
}
```

---

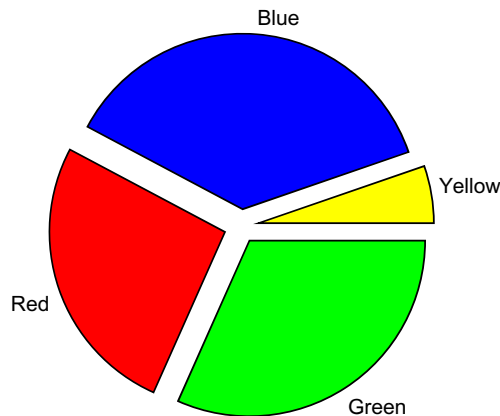
## Pie Chart

A pie chart is a graphical way to organize data. This section describes the construction of a pie chart.

### Example

The `JFrameChart` class is used to create a frame containing a `Chart` node. A `Pie` node is then created as a child of the `Chart` node. The `Pie` node constructor creates `PieSlice` nodes as its children. The number of `PieSlice` nodes created is `y.length`, here equal to 4. After the `PieSlice` nodes are created they are retrieved from the object `pie` and customized by setting attributes.

*Pie Chart*



The `LabelType` attribute is set in the `pie` node. The `pie` node itself does not use this attribute, but from there it is inherited by all of the `PieSlice` nodes.

The `Title` attribute is set in each `PieSlice` node. This is the slice label. It is used to label the slice only if the slice's `LabelType` attribute is `LABEL_TYPE_TITLE`.

The `FillColor` attribute is also set in each slice. This determines the color of the slice. Since the default value of `FillColor` is black, it is generally recommended that `FillColor` be set in each slice.

The `FillOutlineColor` attribute sets the border color of each slice. In this example it is set in the `pie` node to be blue and set in the `slice[1]` node to be yellow. All except `slice[1]` are outlined in blue, and `slice[1]` is outlined in yellow.

The `Explode` attribute moves a pie slice out from the center. Its default value is 0, which puts the slice vertex in the center. A value of 1.0 would put the slice vertex on the circumference.

```
import com.imsl.chart.*;
import java.awt.Color;

public class SamplePieChart extends JFrameChart {

    public SamplePieChart() {
        Chart chart = getChart();

        double y[] = {35., 20., 30., 5.};
        Pie pie = new Pie(chart, y);
        pie.setLabelType(Pie.LABEL_TYPE_TITLE);
        pie.setFillOutlineColor(Color.blue);

        PieSlice[] slice = pie.getPieSlice();

        slice[0].setFillColor(Color.green);
        slice[0].setTitle("Green");
        slice[0].setExplode(0.1);

        slice[1].setFillColor(Color.red);
        slice[1].setTitle("Red");
        slice[1].setFillOutlineColor(Color.yellow);
        slice[1].setExplode(0.1);

        slice[2].setFillColor(Color.blue);
        slice[2].setTitle("Blue");
        slice[2].setExplode(0.1);

        slice[3].setFillColor(Color.yellow);
        slice[3].setTitle("Yellow");
        slice[3].setExplode(0.15);
    }

    public static void main(String argv[]) {
        new SamplePieChart().setVisible(true);
    }
}
```

---

## Box Plot

Box plots are used to show statistics about multiple groups of observations.

For each group of observations, the box limits represent the lower quartile (25-*th* percentile) and upper quartile (75-*th* percentile). The median is displayed as a line across the box. Whiskers are drawn from the upper quartile to the upper adjacent value, and from the lower quartile to the lower adjacent value.

Optional notches may be displayed to show a 95 percent confidence interval about the median, at

$$\pm 1.58 \text{ } IRQ / \sqrt{n}$$

where *IRQ* is the interquartile range and *n* is the number of observations. Outside and far outside values may be displayed as symbols. Outside values are outside the inner fence. Far out values are outside the outer fence.

The BoxPlot has several child nodes. Any of these nodes can be disabled by setting their "Paint" attribute to false.

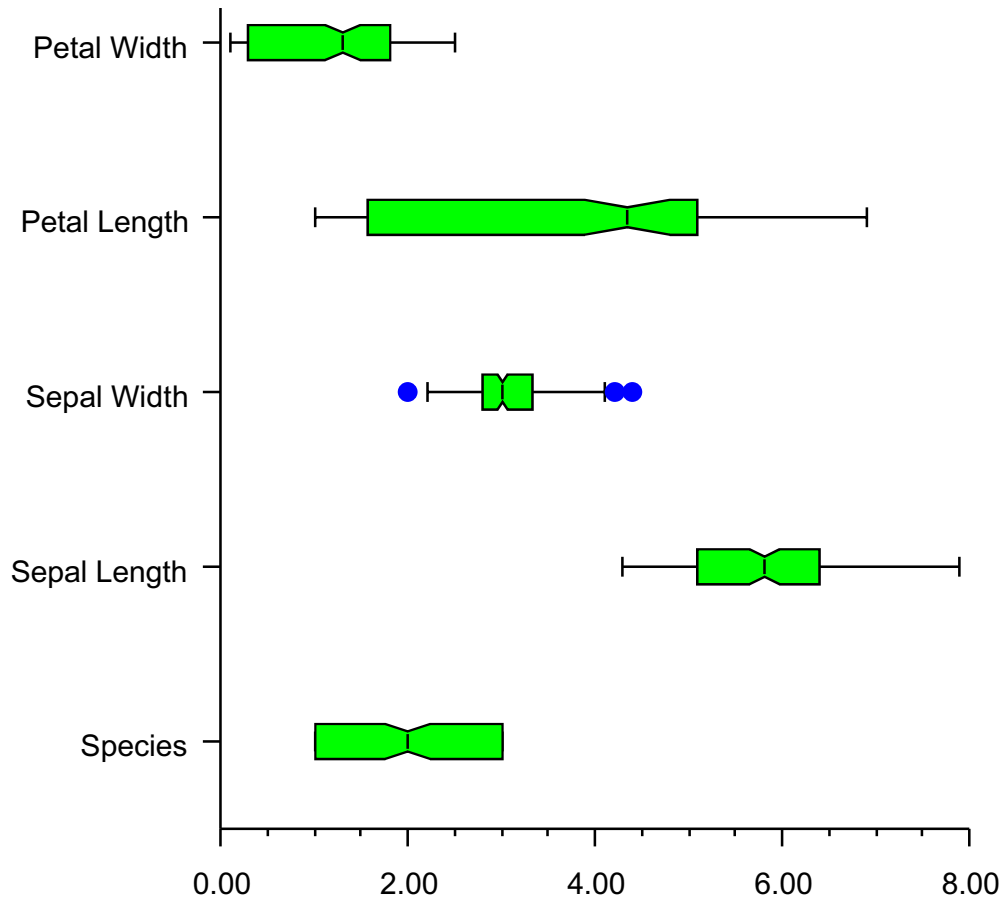
- The "Bodies" node has the main body of the box plot elements. Its fill attributes determine the drawing of (notched) rectangle (see [“Fill Area Attributes” on page 75](#)). Its line attributes determine the drawing of the median line. The width of the box is controlled by the "MarkerSize" attribute (see [“Attribute MarkerSize” on page 74](#)).
- The "Whiskers" node draws the lines to the upper and lower quartile. Its drawing is affected by the marker attributes.
- The "FarMarkers" node hold the far markers. Its drawing is affected by the marker attributes.
- The "OutsideMarkers" node hold the outside markers. Its drawing is affected by the marker attributes.

### Example

In this example, the Fisher iris data set is read from a file and a Box plot is created from data. The data is in a file called "FisherIris.csv" (see [“Attribute MarkerSize” on page 74](#)) in the same directory as this class.

The y-axis labels are taken from the column names.

The boxes are colored green, the markers are all filled circles. The outside markers are blue and the far outside markers would be red, if there were any.





```

import com.imsl.chart.*;
import com.imsl.io.FlatFile;
import java.io.*;
import java.sql.SQLException;
import java.util.StringTokenizer;

public class SampleBoxPlot extends JFrameChart {

    public SampleBoxPlot() throws IOException, java.sql.SQLException {
        // Read the data
        InputStream is =
SampleBoxPlot.class.getResourceAsStream("FisherIris.csv");
        DataReader reader = new DataReader(is);
        int nColumns = 5;
        int nObs = 150;
        String labels[] = new String[nColumns];
        for (int i = 0; i < nColumns; i++) {
            labels[i] = reader.getMetaData().getColumnName(i+1);
        }
        double irisData[][] = new double[nColumns][nObs];
        for (int j = 0; reader.next(); j++) {
            for (int i = 0; i < nColumns; i++) {
                irisData[i][j] = reader.getDouble(i+1);
            }
        }

        // Setup the chart
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        BoxPlot boxPlot = new BoxPlot(axis, irisData);
        boxPlot.setBoxPlotType(BoxPlot.BOXPLOT_TYPE_HORIZONTAL);
        boxPlot.setLabels(labels);
        boxPlot.getBodies().setFillColor("green");
        boxPlot.setMarkerType(BoxPlot.MARKER_TYPE_FILLED_CIRCLE);
        boxPlot.getOutsideMarkers().setMarkerColor("blue");
        boxPlot.getFarMarkers().setMarkerColor("red");
        boxPlot.setNotch(true);
    }

    public static void main(String argv[]) throws IOException, java.sql.SQLEx-
ception {
        new SampleBoxPlot().setVisible(true);
    }

    /**
     * Read the Fisher Iris data

```

```

*/
static private class DataReader extends FlatFile {
    public DataReader(InputStream is) throws IOException {
        super(new BufferedReader(new InputStreamReader(is)));
        String line = readLine();
        StringTokenizer st = new StringTokenizer(line, ",");
        for (int j = 0; st.hasMoreTokens(); j++) {
            setColumnName(j+1, st.nextToken().trim());
            setColumnClass(j, Double.class);
        }
    }
}
}
}

```

```

Species, Sepal Length, Sepal Width, Petal Length, Petal Width
1.0, 5.1, 3.5, 1.4, .2
1.0, 4.9, 3.0, 1.4, .2
1.0, 4.7, 3.2, 1.3, .2
1.0, 4.6, 3.1, 1.5, .2
1.0, 5.0, 3.6, 1.4, .2
1.0, 5.4, 3.9, 1.7, .4
1.0, 4.6, 3.4, 1.4, .3
1.0, 5.0, 3.4, 1.5, .2
1.0, 4.4, 2.9, 1.4, .2
1.0, 4.9, 3.1, 1.5, .1
1.0, 5.4, 3.7, 1.5, .2
1.0, 4.8, 3.4, 1.6, .2
1.0, 4.8, 3.0, 1.4, .1
1.0, 4.3, 3.0, 1.1, .1
1.0, 5.8, 4.0, 1.2, .2
1.0, 5.7, 4.4, 1.5, .4
1.0, 5.4, 3.9, 1.3, .4
1.0, 5.1, 3.5, 1.4, .3
1.0, 5.7, 3.8, 1.7, .3
1.0, 5.1, 3.8, 1.5, .3
1.0, 5.4, 3.4, 1.7, .2
1.0, 5.1, 3.7, 1.5, .4
1.0, 4.6, 3.6, 1.0, .2
1.0, 5.1, 3.3, 1.7, .5
1.0, 4.8, 3.4, 1.9, .2
1.0, 5.0, 3.0, 1.6, .2
1.0, 5.0, 3.4, 1.6, .4
1.0, 5.2, 3.5, 1.5, .2
1.0, 5.2, 3.4, 1.4, .2
1.0, 4.7, 3.2, 1.6, .2
1.0, 4.8, 3.1, 1.6, .2

```

1.0, 5.4, 3.4, 1.5, .4  
1.0, 5.2, 4.1, 1.5, .1  
1.0, 5.5, 4.2, 1.4, .2  
1.0, 4.9, 3.1, 1.5, .2  
1.0, 5.0, 3.2, 1.2, .2  
1.0, 5.5, 3.5, 1.3, .2  
1.0, 4.9, 3.6, 1.4, .1  
1.0, 4.4, 3.0, 1.3, .2  
1.0, 5.1, 3.4, 1.5, .2  
1.0, 5.0, 3.5, 1.3, .3  
1.0, 4.5, 2.3, 1.3, .3  
1.0, 4.4, 3.2, 1.3, .2  
1.0, 5.0, 3.5, 1.6, .6  
1.0, 5.1, 3.8, 1.9, .4  
1.0, 4.8, 3.0, 1.4, .3  
1.0, 5.1, 3.8, 1.6, .2  
1.0, 4.6, 3.2, 1.4, .2  
1.0, 5.3, 3.7, 1.5, .2  
1.0, 5.0, 3.3, 1.4, .2  
2.0, 7.0, 3.2, 4.7, 1.4  
2.0, 6.4, 3.2, 4.5, 1.5  
2.0, 6.9, 3.1, 4.9, 1.5  
2.0, 5.5, 2.3, 4.0, 1.3  
2.0, 6.5, 2.8, 4.6, 1.5  
2.0, 5.7, 2.8, 4.5, 1.3  
2.0, 6.3, 3.3, 4.7, 1.6  
2.0, 4.9, 2.4, 3.3, 1.0  
2.0, 6.6, 2.9, 4.6, 1.3  
2.0, 5.2, 2.7, 3.9, 1.4  
2.0, 5.0, 2.0, 3.5, 1.0  
2.0, 5.9, 3.0, 4.2, 1.5  
2.0, 6.0, 2.2, 4.0, 1.0  
2.0, 6.1, 2.9, 4.7, 1.4  
2.0, 5.6, 2.9, 3.6, 1.3  
2.0, 6.7, 3.1, 4.4, 1.4  
2.0, 5.6, 3.0, 4.5, 1.5  
2.0, 5.8, 2.7, 4.1, 1.0  
2.0, 6.2, 2.2, 4.5, 1.5  
2.0, 5.6, 2.5, 3.9, 1.1  
2.0, 5.9, 3.2, 4.8, 1.8  
2.0, 6.1, 2.8, 4.0, 1.3  
2.0, 6.3, 2.5, 4.9, 1.5  
2.0, 6.1, 2.8, 4.7, 1.2  
2.0, 6.4, 2.9, 4.3, 1.3  
2.0, 6.6, 3.0, 4.4, 1.4

2.0, 6.8, 2.8, 4.8, 1.4  
2.0, 6.7, 3.0, 5.0, 1.7  
2.0, 6.0, 2.9, 4.5, 1.5  
2.0, 5.7, 2.6, 3.5, 1.0  
2.0, 5.5, 2.4, 3.8, 1.1  
2.0, 5.5, 2.4, 3.7, 1.0  
2.0, 5.8, 2.7, 3.9, 1.2  
2.0, 6.0, 2.7, 5.1, 1.6  
2.0, 5.4, 3.0, 4.5, 1.5  
2.0, 6.0, 3.4, 4.5, 1.6  
2.0, 6.7, 3.1, 4.7, 1.5  
2.0, 6.3, 2.3, 4.4, 1.3  
2.0, 5.6, 3.0, 4.1, 1.3  
2.0, 5.5, 2.5, 4.0, 1.3  
2.0, 5.5, 2.6, 4.4, 1.2  
2.0, 6.1, 3.0, 4.6, 1.4  
2.0, 5.8, 2.6, 4.0, 1.2  
2.0, 5.0, 2.3, 3.3, 1.0  
2.0, 5.6, 2.7, 4.2, 1.3  
2.0, 5.7, 3.0, 4.2, 1.2  
2.0, 5.7, 2.9, 4.2, 1.3  
2.0, 6.2, 2.9, 4.3, 1.3  
2.0, 5.1, 2.5, 3.0, 1.1  
2.0, 5.7, 2.8, 4.1, 1.3  
3.0, 6.3, 3.3, 6.0, 2.5  
3.0, 5.8, 2.7, 5.1, 1.9  
3.0, 7.1, 3.0, 5.9, 2.1  
3.0, 6.3, 2.9, 5.6, 1.8  
3.0, 6.5, 3.0, 5.8, 2.2  
3.0, 7.6, 3.0, 6.6, 2.1  
3.0, 4.9, 2.5, 4.5, 1.7  
3.0, 7.3, 2.9, 6.3, 1.8  
3.0, 6.7, 2.5, 5.8, 1.8  
3.0, 7.2, 3.6, 6.1, 2.5  
3.0, 6.5, 3.2, 5.1, 2.0  
3.0, 6.4, 2.7, 5.3, 1.9  
3.0, 6.8, 3.0, 5.5, 2.1  
3.0, 5.7, 2.5, 5.0, 2.0  
3.0, 5.8, 2.8, 5.1, 2.4  
3.0, 6.4, 3.2, 5.3, 2.3  
3.0, 6.5, 3.0, 5.5, 1.8  
3.0, 7.7, 3.8, 6.7, 2.2  
3.0, 7.7, 2.6, 6.9, 2.3  
3.0, 6.0, 2.2, 5.0, 1.5  
3.0, 6.9, 3.2, 5.7, 2.3

3.0, 5.6, 2.8, 4.9, 2.0  
3.0, 7.7, 2.8, 6.7, 2.0  
3.0, 6.3, 2.7, 4.9, 1.8  
3.0, 6.7, 3.3, 5.7, 2.1  
3.0, 7.2, 3.2, 6.0, 1.8  
3.0, 6.2, 2.8, 4.8, 1.8  
3.0, 6.1, 3.0, 4.9, 1.8  
3.0, 6.4, 2.8, 5.6, 2.1  
3.0, 7.2, 3.0, 5.8, 1.6  
3.0, 7.4, 2.8, 6.1, 1.9  
3.0, 7.9, 3.8, 6.4, 2.0  
3.0, 6.4, 2.8, 5.6, 2.2  
3.0, 6.3, 2.8, 5.1, 1.5  
3.0, 6.1, 2.6, 5.6, 1.4  
3.0, 7.7, 3.0, 6.1, 2.3  
3.0, 6.3, 3.4, 5.6, 2.4  
3.0, 6.4, 3.1, 5.5, 1.8  
3.0, 6.0, 3.0, 4.8, 1.8  
3.0, 6.9, 3.1, 5.4, 2.1  
3.0, 6.7, 3.1, 5.6, 2.4  
3.0, 6.9, 3.1, 5.1, 2.3  
3.0, 5.8, 2.7, 5.1, 1.9  
3.0, 6.8, 3.2, 5.9, 2.3  
3.0, 6.7, 3.3, 5.7, 2.5  
3.0, 6.7, 3.0, 5.2, 2.3  
3.0, 6.3, 2.5, 5.0, 1.9  
3.0, 6.5, 3.0, 5.2, 2.0  
3.0, 6.2, 3.4, 5.4, 2.3  
3.0, 5.9, 3.0, 5.1, 1.8

---

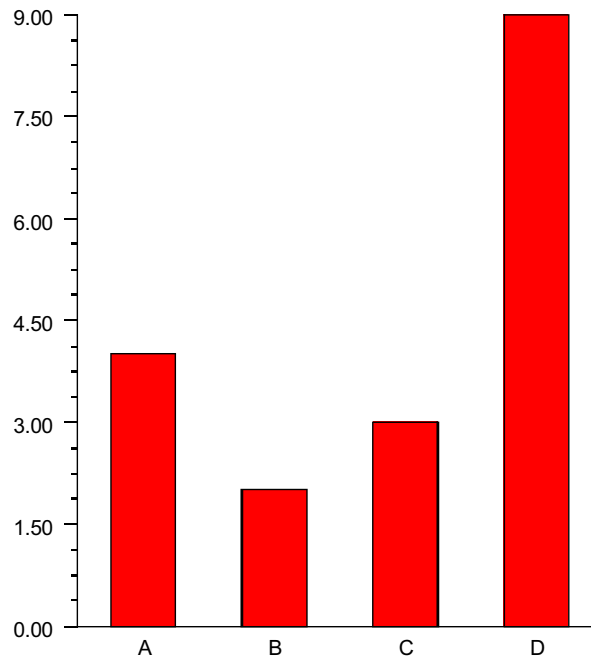
## Bar Chart

The class `Bar` is used to create bar charts and histograms. This section describes the construction of labeled bar charts. For a discussion of histograms, see [“Histogram” on page 64](#).

### Simple Bar Chart

The following code creates this labeled bar chart. The `BarType` attribute can be either `BAR_TYPE_VERTICAL` or `BAR_TYPE_HORIZONTAL`. The method `setLabels` sets the bar labels and adjusts the attributes of the axis to be appropriate for bar labels. The `setLabels` method *must be called after* the `setBarType` method, so that the correct axis has its attributes adjusted.

The drawing of the bars is controlled by the [“FillType” on page 75](#) and [“FillOutlineType” on page 75](#) attributes. By default `FillType` has the value `FILL_TYPE_SOLID`, so setting the associated attribute [“FillColor” on page 75](#) to red causes solid red bars to be drawn.



```
import com.imsl.chart.*;  
import java.awt.Color;
```

```
public class SampleBar extends JFrameChart {

    public SampleBar() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        Bar bar = new Bar(axis, y);
        bar.setBarType(Bar.BAR_TYPE_VERTICAL);
        bar.setLabels(new String[]{"A", "B", "C", "D"});
        bar.setFill(Color.red);
    }

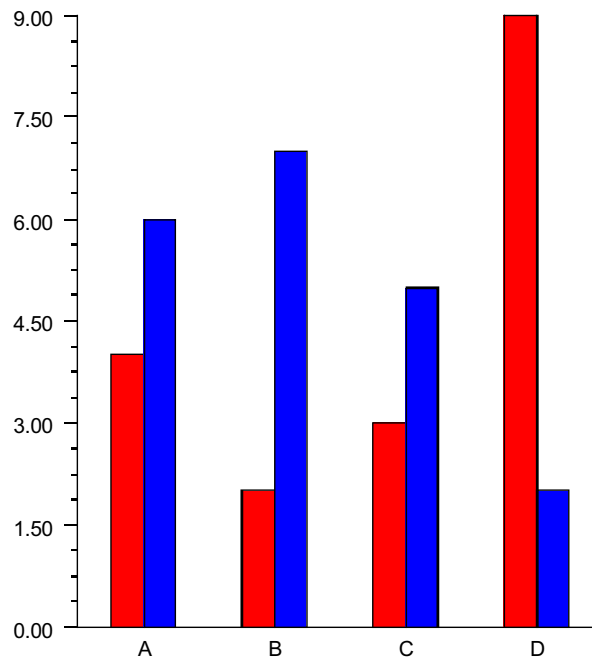
    public static void main(String argv[]) {
        new SampleBar().setVisible(true);
    }
}
```

## Grouped Bar Chart

In a grouped bar chart multiple sets of data are displayed as side-by-side bars.

The data argument to the constructor for a grouped bar chart is an `nGroups` by `nItems` array of doubles. In this example there are two groups, each containing four items. All of the groups must contain the same number of items.

The `getBarSet(int)` method returns a `BarSet` object that is a collection of the `BarItems` that make up a given group. Here the bars in group 0 are set to red and those in group 1 are set to blue.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleBarGroup extends JFrameChart {

    public SampleBarGroup() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[][] = {{4,2,3,9},{6,7,5,2}};
        Bar bar = new Bar(axis, y);
        bar.setBarType(Bar.BAR_TYPE_VERTICAL);
        bar.setLabels(new String[]{"A","B","C","D"});
    }
}
```



```
        bar.getBarSet(0).setFillColor(Color.red);
        bar.getBarSet(1).setFillColor(Color.blue);
    }

    public static void main(String argv[]) {
        new SampleBarGroup().setVisible(true);
    }
}
```

In the above `SampleBarGroup` example, the `Bar` constructor creates a collection of chart nodes. For each group, it creates a `BarSet` node as its direct child. Each `BarSet` node has `BarItem` nodes as children, one for each bar in the set.

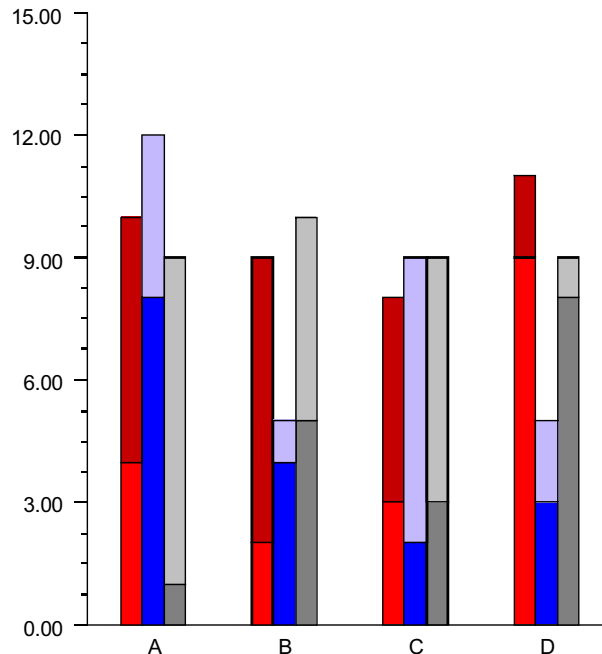
## Stacked Grouped Bar Chart

The most general form of the bar chart is a stacked, grouped bar chart.

The data argument to the constructor for a stacked, grouped bar chart is an `nStacks by nGroups by nItems` array of doubles. In this example there are two stacks in three groups each containing four items. All of the stacks must contain the same number of groups and all of the groups must contain the same number of items.

The `getBarSet(int, int)` method returns a `BarSet` object that is a collection of the `BarItems` that make up a given stack/group. Here within each group the stacks are set to shades of the same color.

A stacked bar chart, without groups, can be constructed as a stacked-grouped bar chart with one group.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleBarGroupStack extends JFrameChart {
    static final Color darkRed = new Color(196,0,0);
    static final Color lightBlue = new Color(196,185,253);
```

```

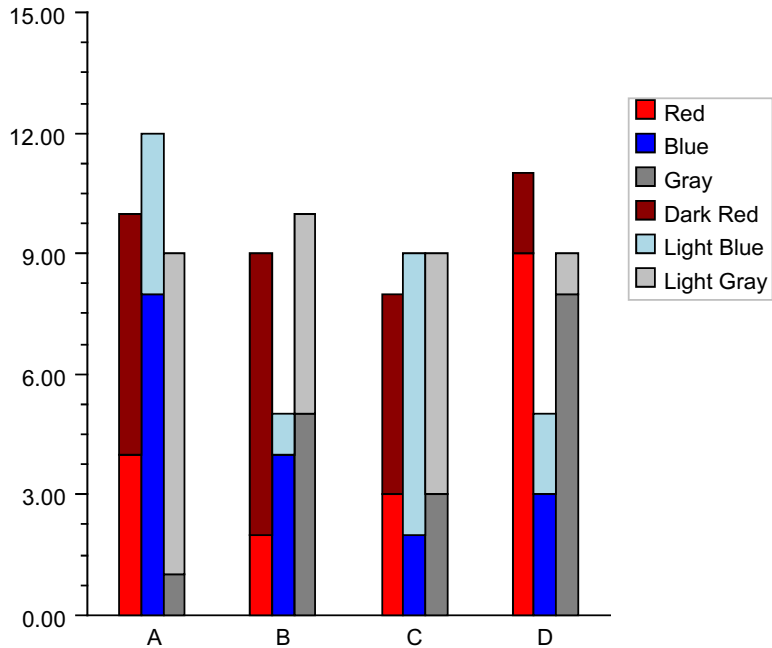
public SampleBarGroupStack() {
    Chart chart = getChart();
    AxisXY axis = new AxisXY(chart);
    // y is a 2 by 3 by 4 array
    double y[][][] = {
        {{4,2,3,9},{8,4,2,3},{1,5,3,8}},
        {{6,7,5,2},{4,1,7,2},{8,5,6,1}}
    };
    Bar bar = new Bar(axis, y);
    bar.setBarType(Bar.BAR_TYPE_VERTICAL);
    bar.setLabels(new String[]{"A","B","C","D"});
    // group 0 - shades of red
    bar.getBarSet(0,0).setFillColor(Color.red);
    bar.getBarSet(1,0).setFillColor(darkRed);
    // group 1 - shades of blue
    bar.getBarSet(0,1).setFillColor(Color.blue);
    bar.getBarSet(1,1).setFillColor(lightBlue);
    // group 2 - shades of gray
    bar.getBarSet(0,2).setFillColor(Color.gray);
    bar.getBarSet(1,2).setFillColor(Color.lightGray);
}

public static void main(String argv[]) {
    new SampleBarGroupStack().setVisible(true);
}
}

```

## Legend

The Legend for a bar chart is turned on by setting the Legend's Paint attribute to `true` and defining the Title attributes for the legend entries. The legend entries are the `BarSet` objects. The following example is the stacked, grouped bar example with the legend enabled.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleBarLegend extends JFrameChart {
    public SampleBarLegend() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        chart.getLegend().setPaint(true);
        // y is a 2 by 3 by 4 array
        double y[][][] = {
            {{4,2,3,9},{8,4,2,3},{1,5,3,8}},
            {{6,7,5,2},{4,1,7,2},{8,5,6,1}}
        };
        Bar bar = new Bar(axis, y);
        bar.setBarType(Bar.BAR_TYPE_VERTICAL);
        bar.setLabels(new String[]{"A","B","C","D"});
        // group 0 - shades of red
        bar.getBarSet(0,0).setTitle("Red");
    }
}
```

```
    bar.getBarSet(0,0).setFillColor("Red");
    bar.getBarSet(1,0).setTitle("Dark Red");
    bar.getBarSet(1,0).setFillColor("DarkRed");
    // group 1 - shades of blue
    bar.getBarSet(0,1).setTitle("Blue");
    bar.getBarSet(0,1).setFillColor("Blue");
    bar.getBarSet(1,1).setTitle("Light Blue");
    bar.getBarSet(1,1).setFillColor("LightBlue");
    // group 2 - shades of gray
    bar.getBarSet(0,2).setTitle("Gray");
    bar.getBarSet(0,2).setFillColor("Gray");
    bar.getBarSet(1,2).setTitle("Light Gray");
    bar.getBarSet(1,2).setFillColor("LightGray");
}

public static void main(String argv[] ) {
    new SampleBarLegend().setVisible(true);
}
}
```

## Attribute BarGap

The BarGap attribute sets the gap between bars in a group. A gap of 1.0 means that space between bars is the same as the width of an individual bar in the group. Its default value is 0.0, meaning there is no space between groups.

## Attribute BarWidth

The BarWidth attribute sets the width of the groups of bars at each index. Its default value is 0.5. If the number of groups is increased, the width of each individual bar is reduced proportionately.

See [“Histogram” on page 64](#) for an example of the use of the BarWidth attribute.

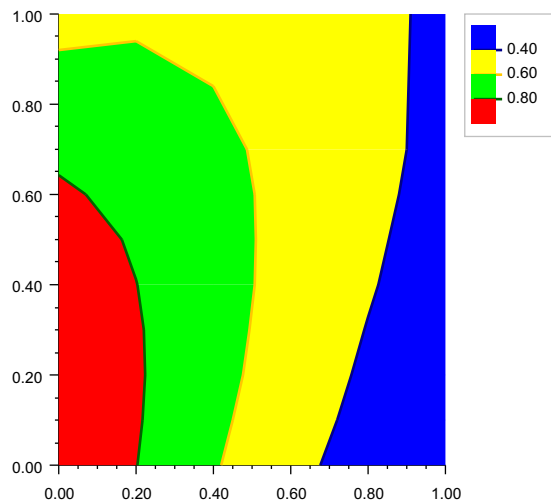
---

## Contour Chart

A Contour chart shows level curves of a two-dimensional function.

### Example

The `JFrameChart` class is used to create a frame containing a `Chart` node. A `Contour` node is then created as a child of the `Chart` node. The `Contour` node constructor creates `ContourLevel` nodes as its children. The number of `ContourLevel` nodes created is equal to the number of contour levels plus one, here equal to 4. After the `ContourLevel` nodes are created they are retrieved from the object pie and customized by setting attributes.



The `FillColor` and `LineColor` attributes are set in each level. This determines the color of the fill area for the level and the color of the level curves. Since the default value of `FillColor` is black, it is generally recommended that `FillColor` be set in each level.

Each level corresponds to the area less than or equal to the contour level value and greater than the previous level, if any. So in this example, since the 0-th level value

is 0.4, the area where the function is less than 0.4 is filled with blue (the level-0 fill color) and the level curve equal to 0.4 is draw with dark blue, the level-0 line color.

```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleContour extends JFrameChart {

    public SampleContour() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double xGrid[] = {0, .2, .4, .6, .8, 1.0};
        double yGrid[] = {0, .1, .2, .3, .4, .5, .6, .7, 1.};
        double zData[][] = new double[xGrid.length][yGrid.length];
        for (int i = 0; i < xGrid.length; i++) {
            for (int j = 0; j < yGrid.length; j++) {
                double x = xGrid[i];
                double y = yGrid[j];
                zData[i][j] = Math.exp(-x)*Math.cos(x-y);
            }
        }
        double level[] = {.4, .6, .8};
        Contour contour = new Contour(axis, xGrid, yGrid, zData, level);
        contour.getContourLegend().setPaint(true);
        contour.getContourLevel(0).setFillColor("blue");
        contour.getContourLevel(1).setFillColor("yellow");
        contour.getContourLevel(2).setFillColor("green");
        contour.getContourLevel(3).setFillColor("red");

        contour.getContourLevel(0).setLineColor("darkblue");
        contour.getContourLevel(1).setLineColor("orange");
        contour.getContourLevel(2).setLineColor("darkgreen");
        contour.getContourLevel(3).setLineColor("darkred");
    }

    public static void main(String argv[]) {
        new SampleContour().setVisible(true);
    }
}
```



---

## Histogram

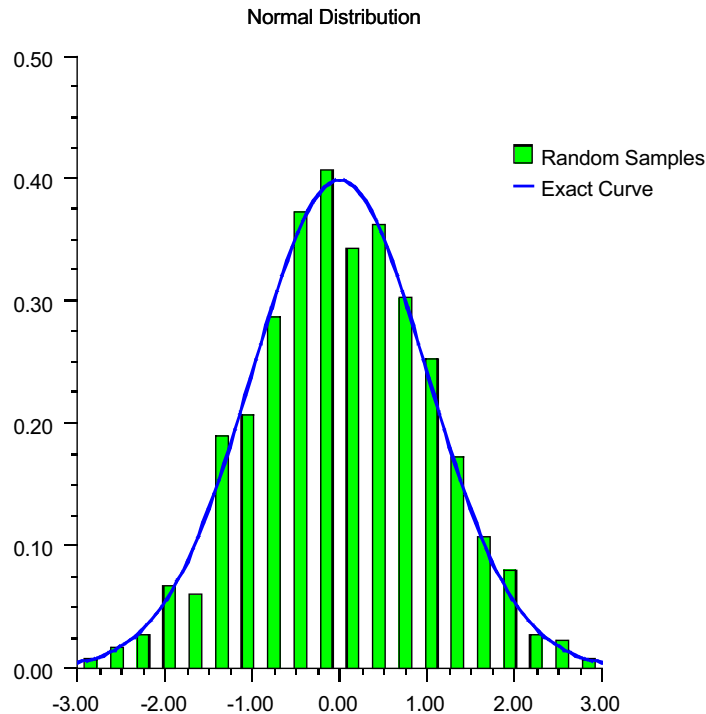
A histogram is a bar chart in which the height of the bars is proportional to the frequencies. A histogram generally uses the same axis style as a scatter plot (i.e. the bars are numbered not labeled.)

In JMSL, histograms are drawn using the `Bar` class, but its `setLabels` method is not used.

### Example

In this example normally distributed random numbers are generated and placed into 20 uniformly sized bins in the interval  $[-3,3]$ . Points outside of this interval are ignored. The bins counts are scaled by the number of samples and the bin width. The scaled bin counts are charted using `Bar` chart. The exact normal distribution is implemented as a `ChartFunction` and plotted.

The legend is displayed by setting the legend node's `Paint` attribute to `true` and defining the bar chart's `Title` attribute. The legend is positioned on the chart by setting its `Viewport` attribute.



```

import com.imsl.math.Sfun;
import com.imsl.stat.Random;
import com.imsl.chart.*;
import java.awt.*;
import javax.swing.*;

public class SampleHistogram extends JFrameChart {

    public SampleHistogram() {
        int nSamples = 1000;
        int nBins = 20;

        // Setup the bins
        double bins[] = new double[nBins];
        double dx = 6.0/nBins;
        double x[] = new double[nBins];
        for (int k = 0; k < nBins; k++) {
            x[k] = -3.0 + (k+0.5)*dx;
        }

        Random r = new Random(123457);
        for (int k = 0; k < nSamples; k++) {
            double t = r.nextNormal();
            int j = (int)Math.round((t+3.0-0.5*dx)/dx);
            if (j >= 0 && j < nBins) bins[j]++;
        }

        // Scale the bins
        for (int k = 0; k < nBins; k++) {
            bins[k] /= nSamples*dx;
        }

        // create the chart
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        chart.getChartTitle().setTitle("Normal Distribution");
        chart.getLegend().setPaint(true);
        chart.getLegend().setViewport(0.7, 1.0, 0.2, 0.3);
        chart.getLegend().setFillOutlineType(chart.FILL_TYPE_NONE);

        Bar bar = new Bar(axis, x, bins);
        bar.setBarType(bar.BAR_TYPE_VERTICAL);
        bar.setFillColor(Color.green);
        bar.setBarWidth(0.5*dx);
        bar.setTitle("Random Samples");

        // plot the expected curve
        ChartFunction f = new ChartFunction() {

```

```
        public double f(double x) {
            return Math.exp(-0.5*x*x)/Math.sqrt(2.0*Math.PI);
        }
};
Data data = new Data(axis, f, -3, 3.0);
data.setLineColor(Color.blue);
data.setTitle("Exact Curve");
data.setLineWidth(2.0);
}

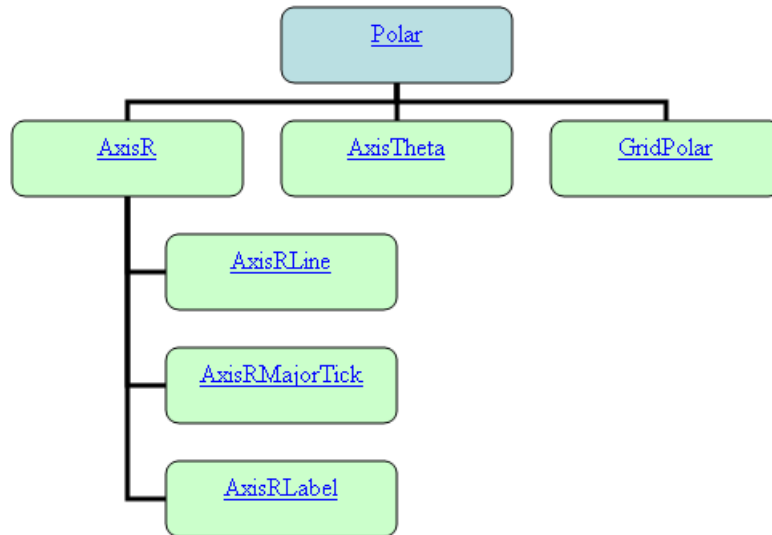
public static void main(String argv[]) {
    new SampleHistogram().setVisible(true);
}
}
```

---

## Polar Plot

The class `Polar` is used to plot  $(r, \text{theta})$  data, where  $r$  is the distance from the origin and  $\text{theta}$  is the angle from the positive  $x$ -axis. Data node  $(x, y)$  values are treated as if they were  $(r, \text{theta})$  values.

When a `Polar` node is created, a subtree of other nodes is created. These automatically created nodes are shown in the following diagram.



`AxisR` is the  $r$ -axis node. This is drawn along the positive  $x$ -axis. This node has three subnodes: `AxisRLine` for the actual line, `AxisRMajorTick` for the tick marks along the line and `AxisRLabel` for the tick mark labels.

The drawing of the `AxisRLine` and the `AxisRMajorTick` is controlled by the [Line Attributes](#) (see [page 72](#)). The drawing of the `AxisRLabel` is controlled by the [Text Attributes](#) (see [page 79](#)).

The `Window` attribute in `AxisR` specifies the length of the  $r$ -axis. Since the  $r$ -axis range is  $[0, r_{max}]$ , the `Window` attribute for this node is the  $r_{max}$  value. Autoscaling is used to automatically determine  $r_{max}$  from the data. Autoscaling can be turned off. See [“Autoscale” on page 89](#) for more details.

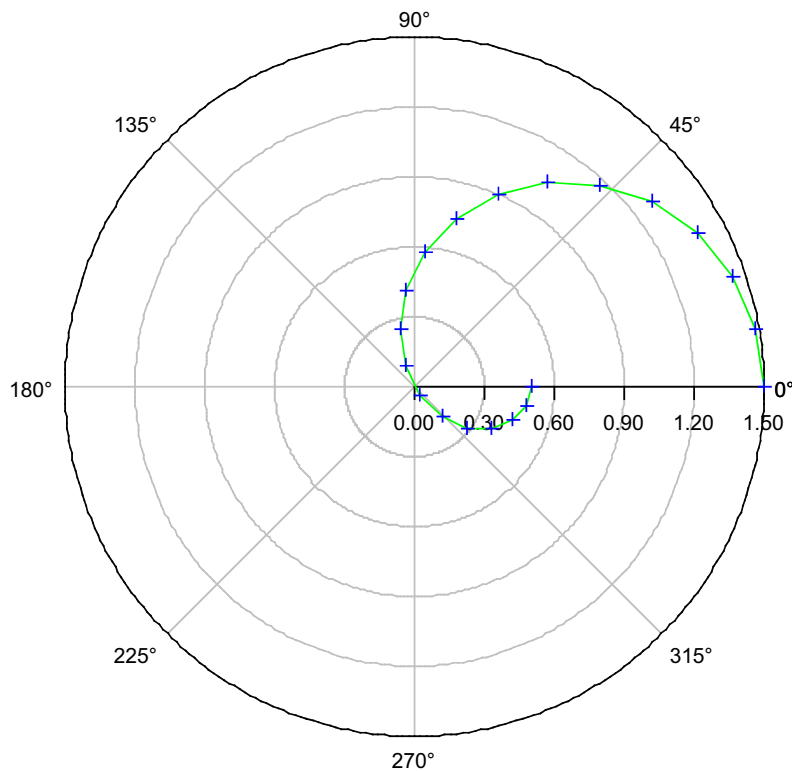
`AxisTheta` is the  $\text{theta}$ -axis node. It is drawn as a circle around the plot. The drawing of the circle itself is controlled by the line attributes. The drawing of the axis labels is controlled by the text attributes.

The `Window` attribute in the `AxisTheta` node contains the angular range, in radians. Its default value is  $[0, 2\pi]$ . The `Window` attribute can be explicitly set to change the range, but autoscaling is *not* available to determine the angular range.

`GridPolar` is the plot grid. It consists of both radial and circular lines. The drawing of these lines is controlled by the line attributes. By default, these lines are drawn and are light gray.

## Example

The function  $r = 0.5 + \cos(\theta)$ , for  $0 \leq \theta \leq \pi$  is plotted.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SamplePolar extends JFrameChart {

    public SamplePolar() {
        Chart chart = getChart();
```

```

Polar axis = new Polar(chart);

double r[] = new double[20];
double theta[] = new double[r.length];
for (int k = 0; k < r.length; k++) {
    theta[k] = Math.PI*k/(r.length-1);
    r[k] = 0.5 + Math.cos(theta[k]);
}
Data data = new Data(axis, r, theta);
data.setDataType(Data.DATA_TYPE_MARKER | Data.DATA_TYPE_LINE);
data.setLineColor(Color.green);
data.setMarkerColor(Color.blue);
}

public static void main(String argv[]) {
    new SamplePolar().setVisible(true);
}
}

```



# Chapter 3: Drawing Elements

This chapter discusses these JMSL chart drawing elements:

- [“Line Attributes” on page 72](#)
- [“Marker Attributes” on page 73](#)
- [“Fill Area Attributes” on page 75](#)
- [“Text Attributes” on page 79](#)
- [“Labels” on page 82](#)
- [“AxisXY” on page 87](#)
- [“Background” on page 119](#)
- [“Legend” on page 125](#)
- [“Tool Tips” on page 129](#)



---

## Line Attributes

All lines, except for those used to draw markers or outline filled regions, are affected by the attributes described in this section. These include axis lines and lines drawn when a Data node is rendered with its `DataType` attribute having its `DATA_TYPE_LINE` bit set.

### Attribute `LineColor`

`LineColor` is a `Color`-valued attribute that determines the color of the line. Its default value is `Color.black`.

### Attribute `LineWidth`

`LineWidth` is a double-valued attribute that determines the thickness of the lines. Its default value is 1.0.

### Attribute `LineDashPattern`

`LineDashPattern` is a double-array-valued attribute that determines the line pattern used to draw the line. It defaults to a solid line. Alternate entries in the array represent the lengths of the opaque and transparent segments of the dashes.

Some dash patterns are defined. They are `DASH_PATTERN_DOT`, `DASH_PATTERN_DASH` and `DASH_PATTERN_DASH_DOT`.

## Samples

————— Normal  
————— LineWidth = 2  
————— LineWidth = 4  
..... LineDashPattern = DASH\_PATTERN\_DOT  
- - - - LineDashPattern = DASH\_PATTERN\_DASH  
- . - . - . LineDashPattern = DASH\_PATTERN\_DASH\_DOT

---

## Marker Attributes

Markers are drawn when a Data node is rendered with its `DataType` attribute having its `DATA_TYPE_MARKER` bit set. Drawing of markers is affected by the attributes described in this section. Note that even though some markers are drawn using lines, the line attributes do not apply to markers.

An alternative to markers are images, which can be used to draw arbitrary symbols instead of markers.

### Attribute `MarkerType`

`MarkerType` is an integer-valued attribute that determines which marker will be drawn. There are constants defined in `ChartNode` for the marker types. The default value is `MARKER_TYPE_PLUS`. The following are all of the defined marker types. For clarity, these are drawn larger than normal.

- \* `MARKER_TYPE_ASTERISK`
- × `MARKER_TYPE_X`
- `MARKER_TYPE_HOLLOW_SQUARE`
- `MARKER_TYPE_FILLED_SQUARE`
- △ `MARKER_TYPE_HOLLOW_TRIANGLE`
- ▲ `MARKER_TYPE_FILLED_TRIANGLE`
- ◇ `MARKER_TYPE_HOLLOW_DIAMOND`
- ◆ `MARKER_TYPE_FILLED_DIAMOND`
- ⊕ `MARKER_TYPE_DIAMOND_PLUS`
- ⊠ `MARKER_TYPE_SQUARE_X`
- ⊞ `MARKER_TYPE_SQUARE_PLUS`
- ⊗ `MARKER_TYPE_OCTAGON_X`
- ⊕ `MARKER_TYPE_OCTAGON_PLUS`
- `MARKER_TYPE_HOLLOW_CIRCLE`
- `MARKER_TYPE_FILLED_CIRCLE`
- ⊗ `MARKER_TYPE_CIRCLE_X`
- ⊕ `MARKER_TYPE_CIRCLE_PLUS`
- ◎ `MARKER_TYPE_CIRCLE_CIRCLE`

## Attribute MarkerColor

MarkerColor is a Color-valued attribute that determines the color of the marker. Its default value is `Color.black`.

## Attribute MarkerSize

MarkerSize is a double-valued attribute that determines the size of the marker. Its default value is 1.0. The actual size of the drawn marker, in pixels, is  $0.007 * \text{MarkerSize} * \text{width}$ , where *width* is the width of the Component containing the chart.

## Attribute MarkerThickness

MarkerThickness is a double-valued attribute that determines the thickness of the lines used to draw the marker. Its default value is 1.0.

## Attribute MarkerDashPattern

MarkerDashPattern is a double-array-valued attribute that determines the line pattern used to draw the marker. It defaults to a solid line. Alternate entries in the array represent the lengths of the opaque and transparent segments of the dashes.

Some dash patterns are defined. They are `DASH_PATTERN_DOT`, `DASH_PATTERN_DASH` and `DASH_PATTERN_DASH_DOT`.

## Samples

- Normal
- MarkerThickness = 2
- MarkerThickness = 4
- MarkerDashPattern = `DASH_PATTERN_DOT`
- MarkerDashPattern = `DASH_PATTERN_DASH`
- MarkerDashPattern = `DASH_PATTERN_DASH_DOT`

---

## Fill Area Attributes

### FillOutlineType

FillOutlineType is an integer-value attribute that turns on or off the drawing of an outline around filled areas. Its value should be `FILL_TYPE_NONE` (for outline off) or `FILL_TYPE_SOLID` (for outline on). The default is to draw solid lines.

### FillOutlineColor

FillOutlineColor is a Color-valued attribute that determines the color used to outline the filled regions. The outline is drawn only if the attribute FillOutlineType has the value `FILL_TYPE_SOLID`. Its default value is `Color.black`.

### FillType

FillType is an integer-value attribute that turns on or off the drawing of the interior of filled areas. Its value should be

- `FILL_TYPE_NONE` for fill off.
- `FILL_TYPE_SOLID` for fill by a single, solid color. This is the default.
- `FILL_TYPE_GRADIENT` for fill by a color gradient.
- `FILL_TYPE_PAINT` for fill by a Paint object. This is usually a texture.

### FillColor

FillColor is a Color-valued attribute that determines the color used to fill a region with a solid color. Its default value is `Color.black`.


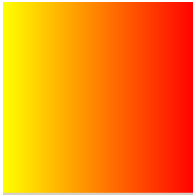


### Gradient

Gradient is a Color array-valued attribute that fills a region with a gradient color. It does not have a default value.

The value of Gradient should be an array of four colors. These are the colors at the four corners of a square. In order they are: lower-left, lower-right, upper-right and upper-left.

- If the lower-side colors are equal (`color[0]` equals `color[1]`) and the upper-side colors are equal (`color[2]` equals `color[3]`), then a vertical gradient is drawn.
- If the left-side colors are equal (`color[0]` equals `color[3]`) and the right-side colors are equal (`color[1]` equals `color[2]`), then a horizontal gradient is drawn.
- If the lower-right and upper-left colors (`color[1]` and `color[3]`) are null, then a diagonal gradient is drawn using the lower-left (`color[0]`) and upper-right (`color[2]`) colors.
- If the lower-left and upper-right colors (`color[0]` and `color[2]`) are null, then a diagonal gradient is drawn using the lower-right (`color[1]`) and upper-left (`color[3]`) colors.

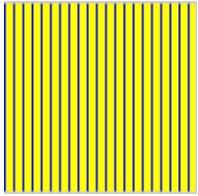



If none of the above patterns exist, then no gradient is drawn.

|            |  |  |
|------------|--|--|
| Vertical   |   | <code>setGradient(Color.yellow, Color.yellow, Color.red, Color.red)</code> |
| Horizontal |   | <code>setGradient(Color.yellow, Color.red, Color.red, Color.yellow)</code> |
| Diagonal   |   | <code>setGradient(Color.yellow, null, Color.red, null)</code>              |
| Diagonal   |  | <code>setGradient(null, Color.yellow, null, Color.red)</code>              |

## FillPaint

FillPaint is a Paint-valued attribute that fills a region with a tiled pattern. It does not have a default value. The class FillPaint contains utilities to define some useful paint patterns.

### Some Examples of FillPaint

|   |   |
|---|---|
|    | <code>FillPaint.verticalStripe(10, 5, Color.yellow,Color.blue)</code>   |
|    | <code>FillPaint.horizontalStripe(10, 5, Color.yellow,Color.blue)</code> |
|   | <code>FillPaint.diamond(36, 5, Color.blue,Color.yellow)</code>          |
|  | <code>FillPaint.checkerboard(24, Color.red,Color.yellow)</code>         |

The FillPaint attribute can also be set using an Image, which is used to tile filled regions

---

## Text Attributes

### Attribute Font

Text is drawn using Font objects constructed using the `FontName`, `FontSize` and `FontStyle` attributes. The `ChartNode.setFont(Font)` method does not save the Font object, but sets these three attributes.

This arrangement allows one to specify font size and style at lower nodes and change the font face at the root node.

Multiline text strings are allowed. They are created by newline characters in the string that creates the text item.

### Attribute FontName

`FontName` is a string-valued attribute that specifies the logical font name or a font face name. Its default value is "SansSerif". Java always defines the font names "Dialog", "DialogInput", "Monospaced", "Serif", "SansSerif", and "Symbol". Depending on the system, additional font names may be defined.

### Attribute FontSize

`FontSize` is an integer-valued attribute that specifies the point size of the font. Its default value is 12.

### Attribute FontStyle

`FontStyle` is an integer-valued attribute that specifies the font style. This can be a bitwise combination of `Font.PLAIN`, `Font.BOLD` and/or `Font.ITALIC`. Its default value is `Font.PLAIN`.



## Samples

Following are sample fonts in styles plain, italic and bold. These may look different on different platforms.

|             |                    |                    |
|-------------|--------------------|--------------------|
| Dialog      | <i>Dialog</i>      | <b>Dialog</b>      |
| DialogInput | <i>DialogInput</i> | <b>DialogInput</b> |
| Monospaced  | <i>Monospaced</i>  | <b>Monospaced</b>  |
| Serif       | <i>Serif</i>       | <b>Serif</b>       |
| SansSerif   | <i>SansSerif</i>   | <b>SansSerif</b>   |

## Attribute TextAngle

TextAngle is an integer-valued attribute that specifies the angle, in degrees, at which text is drawn. The angle is measured counterclockwise. Its default value is 0.

## Attribute TextColor

TextColor is a Color-valued attribute that specifies the color in which the text is drawn. Its default value is `Color.black`.

## Attribute TextFormat

TextFormat is a `java.text.Format`-valued or a String-valued attribute that specifies how to format string objects.

TextFormat can also be set using a String, which is used to generate a Format object when the attribute is accessed. The Format object is created using the value of locale in the chart node.

Some strings have special meanings (case is ignored in these strings):

- "Date(SHORT)" means use:  
`DateFormat.getDateInstance(SHORT, locale)`
- "Date(MEDIUM)" means use  
`DateFormat.getDateInstance(MEDIUM, locale)`
- "Date(LONG)" means use:  
`DateFormat.getDateInstance(LONG, locale)`
- "Currency" means use  
`NumberFormat.getCurrencyInstance(locale)`
- "Percent" means use:  
`NumberFormat.getPercentInstance(locale)`

If `TextFormat` has a string value that is not one of the above, then

```
new DecimalFormat(value, new DecimalFormatSymbols(locale))
```

is used. For example, if its value is "0.00", then numbers will be formatted with exactly two digits after the decimal place. See `DecimalFormat` for a detailed description of these format patterns.

The default value of `TextFormat` is the value returned by the factory method `NumberFormat.getInstance(locale)`.

## Attribute Title

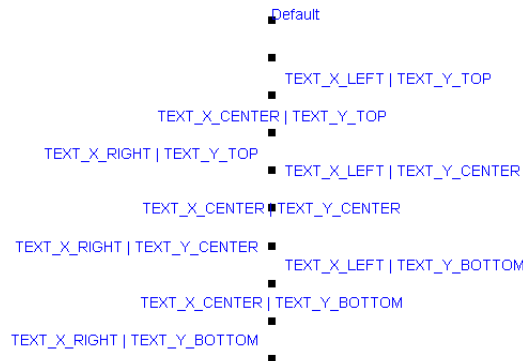
Title is a `Text`-valued attribute that contains the title for a node. The class `Text` holds a string and its alignment and offset information.

The alignment of a `Text` object is the bitwise combination of one of

- `TEXT_X_LEFT`, `TEXT_X_CENTER`, `TEXT_X_RIGHT`, and one of
- `TEXT_Y_BOTTOM`, `TEXT_Y_CENTER`, `TEXT_Y_TOP`.

The offset moves the start of the text away from the reference point in the direction of the alignment. So if the alignment bit `TEXT_X_LEFT` is set and the offset is greater than zero then the text starts a distance further to the left than if the offset were zero. The distance moved is the value of offset times the default marker size. The offset is usually zero, but the `Data` node sets it to 2.0 for labeling data points.

A `Text` object is drawn relative to a reference point. The alignment specifies the position of the reference point on the box that contains the text. There are nine such possible positions. In the following samples, the reference point is marked with a square



If the text is drawn at an angle, then the alignment is relative to the horizontally/vertically aligned bounding box of the text.

---

## Labels

Labels annotate data points or pie slices. As a degenerate case they can be used to place text on a chart without drawing a point. Labels are controlled by the value of the attribute `LabelType` in a `Data` node.

Multiline labels are allowed. They are created by newline characters in the string that creates the label.

### Attribute `LabelType`

The attribute `LabelType` takes on one of the following values.

- `LABEL_TYPE_NONE` for no label. This is the default.
- `LABEL_TYPE_X` label with the value of the  $x$ -coordinate.
- `LABEL_TYPE_Y` label with the value of the  $y$ -coordinate.
- `LABEL_TYPE_TITLE` label with the value of the `Title` attribute.
- `LABEL_TYPE_PERCENT` label with the percentage value. This applies only to `PieSlice` objects.

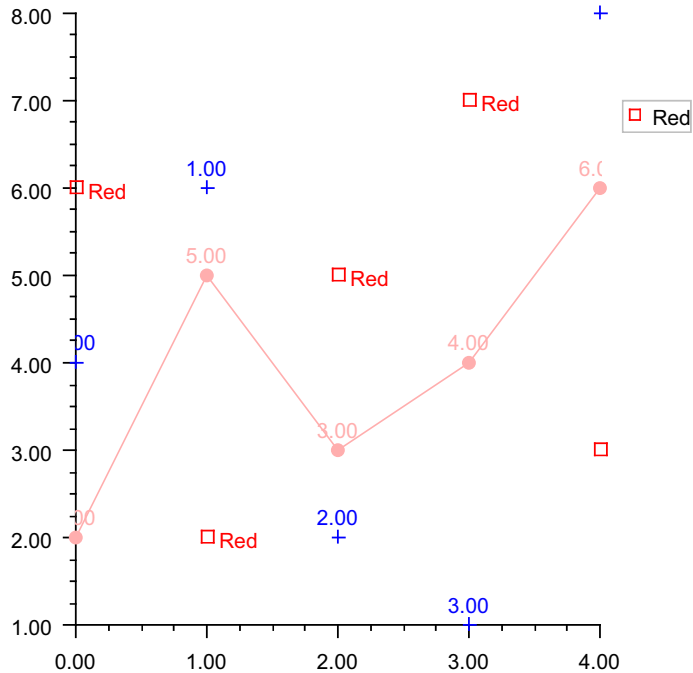
### Data Point Labeling Example

This example shows three data sets, each labeled in a different fashion.

The first data set is blue and the points are labeled by the value of their  $x$ -coordinates.

The second data set is pink and the points are labeled by the value of their  $y$ -coordinates. The data set is drawn with both markers and lines.

The third data set is red and the points are labeled with the `Data` node's `Title` attribute.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleLabels extends JFrameChart {

    public SampleLabels() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        chart.getLegend().setPaint(true);

        // Data set 1 - labeled with X
        double y1[] = {4, 6, 2, 1, 8};
        Data data1 = new Data(axis, y1);
        data1.setDataType(Data.DATA_TYPE_MARKER);
        data1.setMarkerColor(Color.blue);
        data1.setTextColor(Color.blue);
        data1.setLabelType(Data.LABEL_TYPE_X);

        // Data set 2 - labeled with Y
        double y2[] = {2, 5, 3, 4, 6};
        Data data2 = new Data(axis, y2);
```

```

data2.setDataType(Data.DATA_TYPE_MARKER | Data.DATA_TYPE_LINE);
data2.setMarkerColor(Color.pink);
data2.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
data2.setLineColor(Color.pink);
data2.setTextColor(Color.pink);
data2.setLabelType(Data.LABEL_TYPE_Y);

// Data set 3 - labeled with Title
double y3[] = {6, 2, 5, 7, 3};
Data data3 = new Data(axis, y3);
data3.setDataType(Data.DATA_TYPE_MARKER);
data3.setMarkerColor(Color.red);
data3.setMarkerType(Data.MARKER_TYPE_HOLLOW_SQUARE);
data3.setLineColor(Color.red);
data3.setTextColor(Color.red);
data3.setLabelType(Data.LABEL_TYPE_TITLE);
data3.setTitle("Red");
data3.getTitle().setAlignment(Data.TEXT_X_LEFT |
Data.TEXT_Y_CENTER);
}

public static void main(String argv[]) {
    new SampleLabels().setVisible(true);
}
}

```

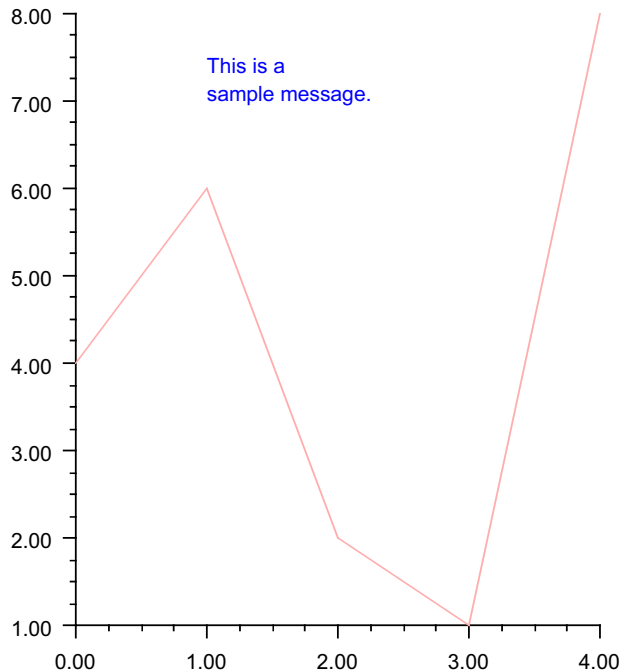
## Annotation

Labels can be used to place text on a chart, without an associated point being drawn. In this example a text message is drawn at (1,7) as its bottom left corner.

Note that it is a two line message. Newlines in the string are reflected in the chart.

The label offset is set to zero, so that the message will be placed exactly at (1,7). This overrides the default offset of 2.0 applied to data labels. The text alignment is set to `TEXT_X_LEFT | TEXT_Y_BOTTOM`, so (1,7) is the lower left corner of the text's bounding box (see [“Attribute Title” on page 81](#)).

No marker or line is drawn because the `DataType` attribute has its default value of zero.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleLabelMessage extends JFrameChart {

    public SampleLabelMessage() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
```

```

double y[] = {4, 6, 2, 1, 8};
Data data = new Data(axis, y);
data.setDataType(Data.DATA_TYPE_LINE);
data.setLineColor(Color.pink);

double xLabel[] = {1.0};
double yLabel[] = {7.0};
Data dataMessage = new Data(axis, xLabel, yLabel);
dataMessage.setTextColor(Color.blue);
dataMessage.setLabelType(Data.LABEL_TYPE_TITLE);
dataMessage.setTitle("This is a\nsample message.");
dataMessage.getTitle().setAlignment(Data.TEXT_X_LEFT |
Data.TEXT_Y_BOTTOM);
    dataMessage.getTitle().setOffset(0.0);
}

public static void main(String argv[]) {
    new SampleLabelMessage().setVisible(true);
}
}

```

---

## AxisXY

The `AxisXY` node is the basis of many chart types, including scatter, line, area and error bar. Its parent node must be the root `Chart` node.

When an `AxisXY` node is created, it creates two `Axis1D` nodes. They can be obtained by using the methods `AxisXY.GetAxisX()` and `AxisXY.GetAxisY()`. Each of the `Axis1D` nodes in turn creates additional child nodes, as seen in the diagram below.

Accessor methods can be chained together, so the x-axis line can be retrieved using

```
axis.GetAxisX().getAxisLine()
```

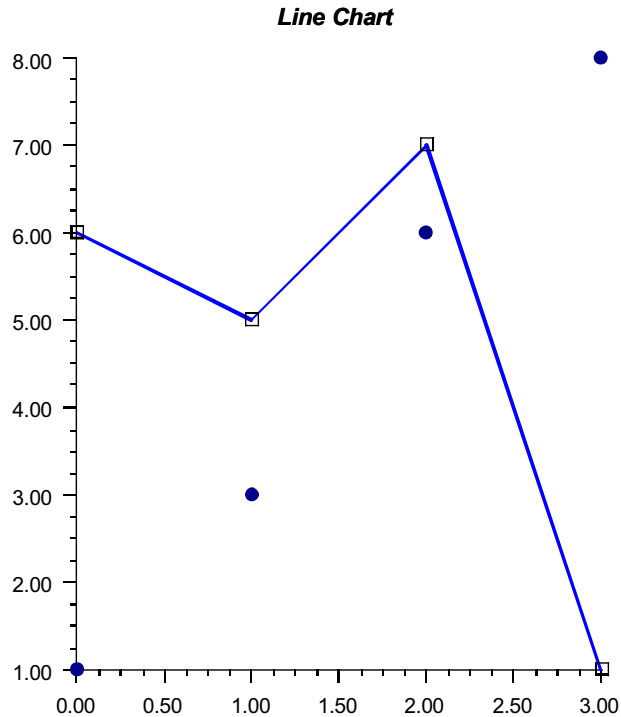
The code to set the x-axis line to blue is

```
axis.GetAxisX().getAxisLine().setLineColor(Color.blue)
```



## Axis Layout

The layout of an AxisXY chart, such as the one below, is controlled by the attributes Window, Density, and Number.



### Window

Window is a double-array-valued attribute that contains the axis limits. Its value is  $\{min, max\}$ . For the above chart, the value of the Window attribute for the  $x$ -axis is  $[0, 3]$  and for the  $y$ -axis it is  $[1, 8]$ .

### Number

Number is an integer-valued attribute that contains the number of major tick marks along an axis. In an Axis1D node its default value is 5, which is also the value in the above example.

### Density

Density is the number of minor tick marks per major tick mark. Its default value is 4, which is also the value in the example.

## Transform

The  $x$ - and  $y$ -axes may be linear, logarithmic, or customized, as specified by the Transform attribute. This attribute can have the values:

- TRANSFORM\_LINEAR indicating a linear axis. This is the default.
- TRANSFORM\_LOG indicating a logarithmic axis
- TRANSFORM\_CUSTOM indicating a custom transformation. The transformation specified by the CustomTransform attribute is used.

## CustomTransform

A customized transformation is used when the Transform attribute has the value TRANSFORM\_CUSTOM. A custom transform is an object implementing the Transform interface. It defines a mapping from the interval specified by the Window attribute to and from the interval  $[0, 1]$ . See [“Custom Transform” on page 110](#) for more information.

The class TransformDate implements the Transform interface. It maps from a regular calendar to one without weekends, which is useful for charting stock prices that are defined only on weekdays.

## Autoscale

Autoscaling is used to automatically determine the attribute Window (the range of numbers along an axis) and the attribute Number (the number of major tick marks). The goal is to adjust the attributes so that the data fits on the axes and the axes have “nice” numbers as labels.

Autoscaling is done in two phases. In the first (“input”) phase the actual range is determined. In the second (“output”) phase chart attributes are updated.

## Attribute AutoscaleInput

The action of the input phase is controlled by the value of attribute AUTOSCALE\_INPUT in the axis node. It can have one of three values.

- AUTOSCALE\_OFF turns off autoscaling.
- AUTOSCALE\_DATA scans the values in the Data nodes that are attached to the axis to determine the data range. This is the default value.
- AUTOSCALE\_WINDOW uses the value of the Window attribute to determine the data range.

## Attribute AutoscaleOutput

The value of the `AUTOSCALE_OUTPUT` attribute can be the bitwise combination of the following values.

- `AUTOSCALE_OFF` no attributes updated.
- `AUTOSCALE_NUMBER` updates the value of the Number attribute. This is the number of major tick marks along the axis.
- `AUTOSCALE_WINDOW` updates the value of the Window attribute. This is the range of numbers displayed along the axis.

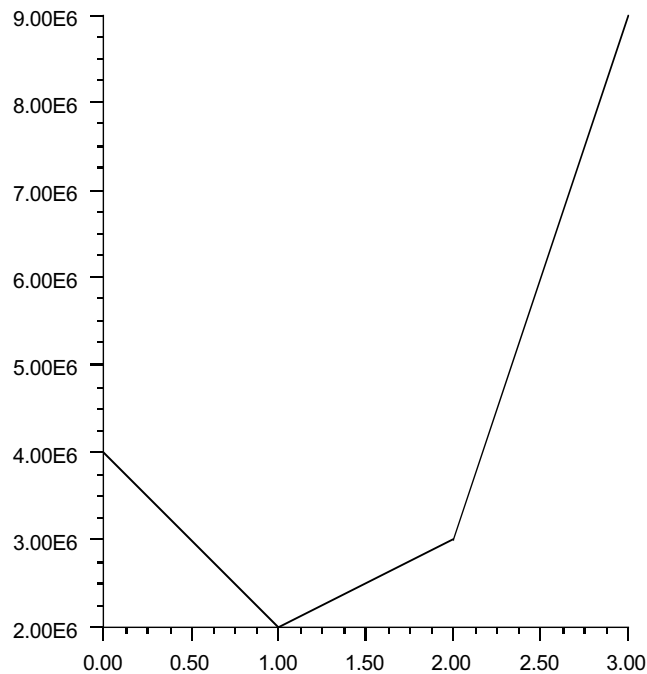
The default is `AUTOSCALE_NUMBER | AUTOSCALE_WINDOW`; both the attributes Number and Window are adjusted.

## Axis Label

The `AxisLabel` node controls the labeling of an axis. The drawing of this node is controlled by the [Text Attributes](#) (see [page 79](#)).

### Scientific Notation

If the values along an axis are large, scientific notation is more readable than a decimal format with many zeros. In this example the y-axis is labeled with scientific notation where each number has exactly two fractional digits displayed. This format pattern is `"0.00E0"`. See `java.text.DecimalFormat` for details on number formatting patterns.



```
import com.imsl.chart.*;

public class SampleAxisLabel1 extends JFrameChart {

    public SampleAxisLabel1() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        axis.getAxisY().getAxisLabel().setTextFormat("0.00E0");
    }
}
```

```

        double y[] = {4e6, 2e6, 3e6, 9e6};
        new Data(axis, y);
    }

    public static void main(String argv[]) {
        new SampleAxisLabel1().setVisible(true);
    }
}

```

## Date Labels

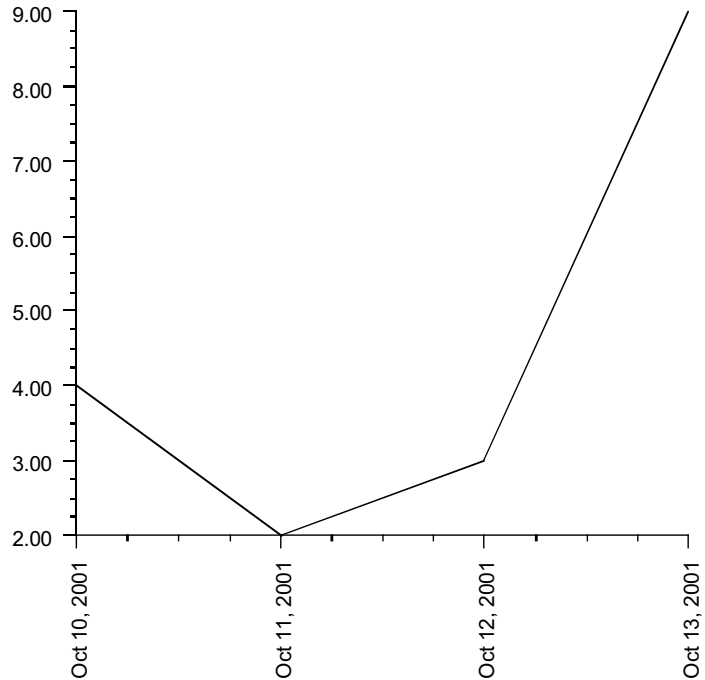
If the `TextFormat` attribute for an axis is an instance of `DateFormat`, then the axis is scaled and labeled as a date/time axis, instead of as a real axis.

Date information passed to the `Date` constructor must be a double number representing the number of milliseconds since the standard base time known as “the epoch”, namely January 1, 1970, 00:00:00 GMT. This is used by the constructor for `Date`.

In this example the  $x$ -axis is labeled with dates. The `setTextFormat(String)` method translates the string `"Date(MEDIUM)"` into a date format string correct for the current locale (here U.S. English). If more control is desired, the method `setTextFormat(Format)` could be used with any `Format` object.

To make room for the text, the labels are rotated 90 degrees. Also, the viewport is changed to make more room for the  $x$ -axis labels.

Since the  $x$ -axis label `TextFormat` is a `DateFormat`, date/time autoscaling is used on the  $x$ -axis. The default real number autoscaling is still used on the  $y$ -axis.



```

import com.imsl.chart.*;
import java.util.Date;
import java.util.GregorianCalendar;

public class SampleAxisLabelDate extends JFrameChart {

    public SampleAxisLabelDate() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        axis.getAxisX().getAxisLabel().setTextFormat("Date(MEDIUM)");
        axis.getAxisX().getAxisLabel().setTextAngle(90);

        GregorianCalendar t[] = {
            new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 10),
            new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 11),
            new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 12),
            new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 13)
        };
        double x[] = new double[4];
        for (int k = 0; k < x.length; k++) x[k] = t[k].getTime().getTime();
        double y[] = {4, 2, 3, 9};
        new Data(axis, x, y);
    }
}

```

```
        axis.setViewport(0.2, 0.9, 0.1, 0.7);
    }

    public static void main(String argv[]) {
        new SampleAxisLabelDate().setVisible(true);
    }
}
```

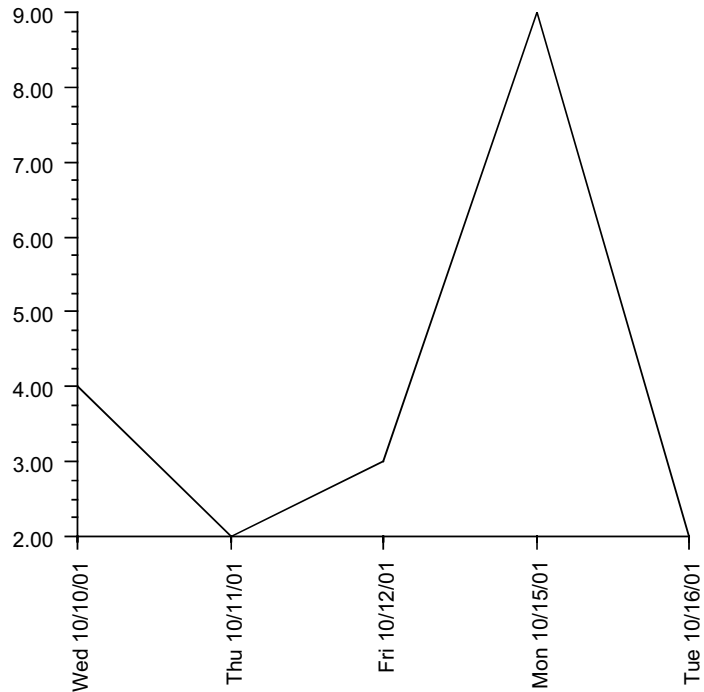
## Skipping Weekends

An additional feature of Date axes is the ability to skip weekends. This feature is often needed for charting stock price data.

To skip weekends it is necessary to adjust the autoscaling for weekdays-only. This is done by setting the attribute `SkipWeekends` to `true`. It is also necessary to set a custom transformation, `TransformDate`, on the axis. This is shown in the following example, which is a modification of the example in the previous section.

Skipping weekends is intended only for data sets where no weekend data exists. It will not give the expected results if the data set contains weekend data.

This example also uses a custom `SimpleDateFormat` object to include the day of the week in the label.



```

import com.imsl.chart.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;

public class SampleWeekend extends JFrameChart {

    public SampleWeekend() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        Axis1D axisX = axis.getAxisX();
        axisX.getAxisLabel().setTextFormat(new SimpleDateFormat("EEE
MM/d/yy"));
        axisX.getAxisLabel().setTextAngle(90);

        // Skip weekends
        axisX.setSkipWeekends(true);
    }
}

```



```

axisX.setTransform(axisX.TRANSFORM_CUSTOM);
axisX.setCustomTransform(new TransformDate());

GregorianCalendar t[] = {
    new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 10),
    new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 11),
    new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 12),
    new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 15),
    new GregorianCalendar(2001, GregorianCalendar.OCTOBER, 16)
};
double x[] = new double[5];
for (int k = 0; k < x.length; k++) x[k] =
t[k].getTime().getTime();
double y[] = {4, 2, 3, 9, 2};
new Data(axis, x, y);

axis.setViewport(0.2, 0.9, 0.1, 0.7);
}

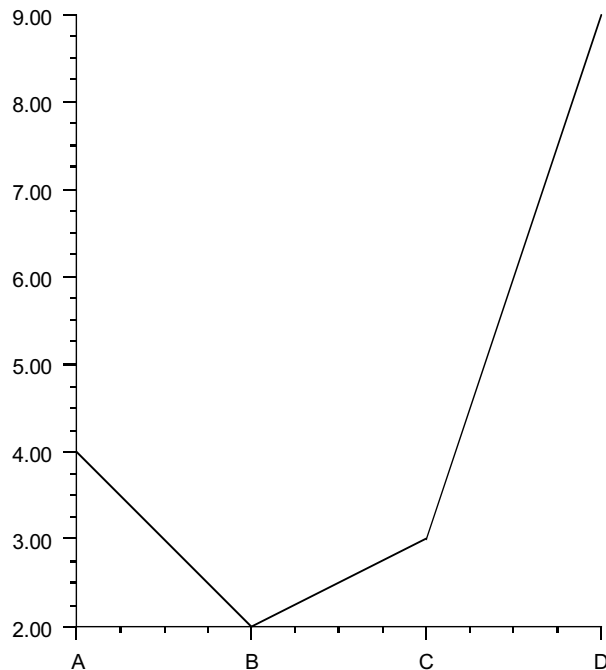
public static void main(String argv[]) {
    new SampleWeekend().setVisible(true);
}
}

```

## String Labels

In this example the program specifies the strings to be used as the *x*-axis labels. The strings are specified using the `setLabels(String[])` method.

The strings are used as labels at the major tick marks. The `setLabels(String[])` method sets the `Number` attribute to the number of strings. In the example the `AutoscaleOutput` attribute is set to `AUTOSCALE_WINDOW`. Its default value is `AUTOSCALE_WINDOW | AUTOSCALE_NUMBER`, so the effect is to keep autoscaling to set the window range, but not to change the number of major tick marks.



```
import com.imsl.chart.*;

public class SampleAxisLabelStrings extends JFrameChart {

    public SampleAxisLabelStrings() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);

        String labels[] = {"A", "B", "C", "D"};
        axis.getAxisX().getAxisLabel().setLabels(labels);
    }
}
```

```
        axis.getAxisX().setAutoscaleOutput(ChartNode.AUTOSCALE_WINDOW);
    }

    public static void main(String argv[]) {
        new SampleAxisLabelStrings().setVisible(true);
    }
}
```

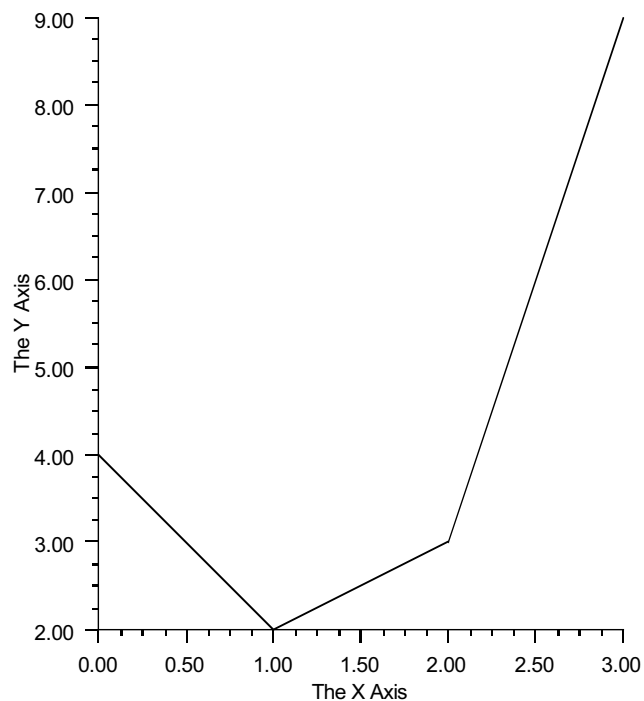
## Axis Title

The AxisTitle node controls the titling of an axis. The drawing of this node is controlled by the [Text Attributes](#) (see [page 79](#)).

Axes are titled with the value of the Title attribute in the AxisTitle node. By default, the title is empty.

## Setting the Title

To set an axis title, set the Title attribute in the AxisTitle node. In this example both the *x*-axis and *y*-axis titles are set.



```
import com.imsl.chart.*;

public class SampleAxisTitle extends JFrameChart {

    public SampleAxisTitle() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);
    }
}
```

```
        axis.getAxisX().getAxisTitle().setTitle("The X Axis");
        axis.getAxisY().getAxisTitle().setTitle("The Y Axis");
    }

    public static void main(String argv[]) {
        new SampleAxisTitle().setVisible(true);
    }
}
```

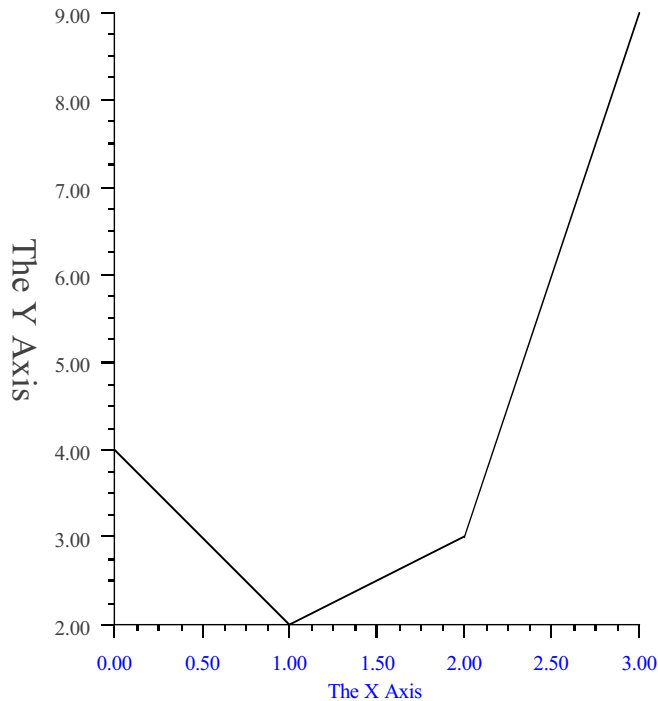
## More Formatting

An axis title, like any other text string, can have further formatting applied.

The `FontName` attribute is set to "Serif" in the axis node. This value is then inherited by all of its ancestor nodes.

The `TextColor` attribute is set differently in the *x*-axis and *y*-axis nodes. Note that this setting is inherited by both the `AxisTitle` and nodes within the same axis.

The *y*-axis title node has its `FontSize` attribute set to 20, so it is larger. Also, its `TextAngle` attribute is set to -90. By default, the *y*-axis title is drawn at a 90 degree angle, so the -90 degree setting flips the title from its usual position.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleAxisTitleFormatted extends JFrameChart {

    public SampleAxisTitleFormatted() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
    }
}
```

```
new Data(axis, y);

axis.setFontName("Serif");

axis.getAxisX().setTextColor(Color.blue);
axis.getAxisX().getAxisTitle().setTitle("The X Axis");

axis.getAxisY().setTextColor(Color.darkGray);
axis.getAxisY().getAxisTitle().setTitle("The Y Axis");
axis.getAxisY().getAxisTitle().setTextAngle(-90);
axis.getAxisY().getAxisTitle().setFontSize(20);
}

public static void main(String argv[]) {
    new SampleAxisTitleFormatted().setVisible(true);
}
}
```

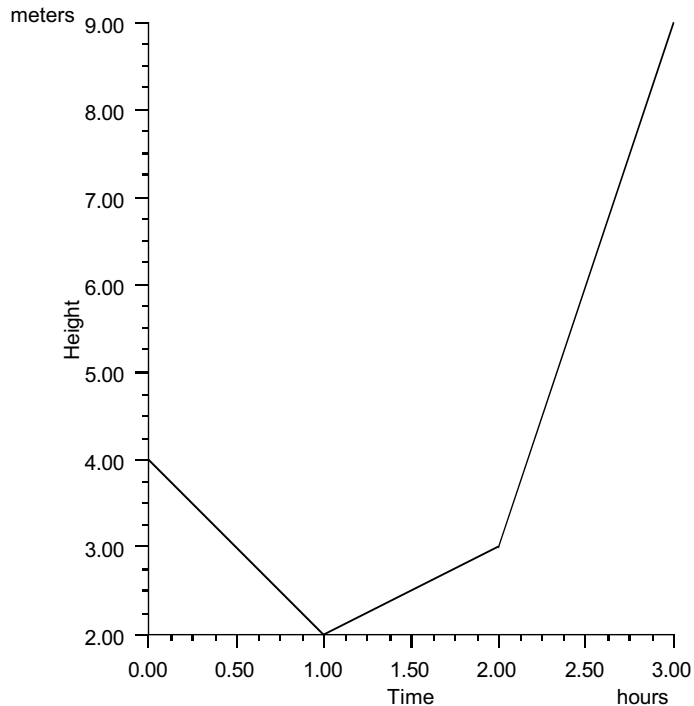
## Axis Unit

The AxisUnit node controls placing of a unit tag on an axis. The tag is the value of the Title attribute in the node. By default, it is empty.

The axis unit is used for labeling only; it has no other effect on the chart.

### Setting the Axis Unit

In this example, the axis unit for the *x*-axis is set to "hours" and the axis unit for the *y*-axis is set to "meters". While not required, the axis titles are also set.



```
import com.imsl.chart.*;

public class SampleAxisUnit extends JFrameChart {

    public SampleAxisUnit() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);

        axis.getAxisX().getAxisTitle().setTitle("Time");
        axis.getAxisY().getAxisTitle().setTitle("Height");
    }
}
```



```

axis.getAxisX().getAxisUnit().setTitle("hours");
axis.getAxisY().getAxisUnit().setTitle("meters");
}

public static void main(String argv[]) {
    new SampleAxisUnit().setVisible(true);
}
}

```

## Major and Minor Tick Marks

The nodes `MajorTick` and `MinorTick` control the drawing of tick marks on an `AxisXY`.

The location of the major tick marks can be set explicitly, using the `Axis1D.setTicks(double[])` method. However, it is usually easier to allow autoscaling to automatically set the tick locations (see [“Autoscale” on page 89](#)).

### Attribute `TickLength`

The length of the tick marks is proportional to the screen size. They can be made relatively longer or shorter by setting the attribute `TickLength`. Its default value is 1.0. If its value is negative, the tick marks are drawn in the opposite direction: i.e., into the center of the plot region rather than away from it.

### Attribute `Number`

`Number` is the number of major tick marks along an axis.

### Attribute `Density`

`Density` is the number of minor tick marks in each interval between major tick marks. The minor ticks are equally spaced in user coordinates. If the `Transform` attribute is not `TRANSFORM_LINEAR`, then they will not be equally spaced on the screen.

### Attribute `FirstTick`

The `FirstTick` attribute, in an `Axis1D` node, is the position of the first major tick mark. The default value is the 0-th element of the `Windows` attribute.

### **Attribute TickInterval**

The TickInterval attribute, in an Axis1D node, is the interval between tick marks in the user coordinates. If this attribute is not explicitly set, its value is computed from the attributes Number, Window and Transform.

### **Attribute Ticks**

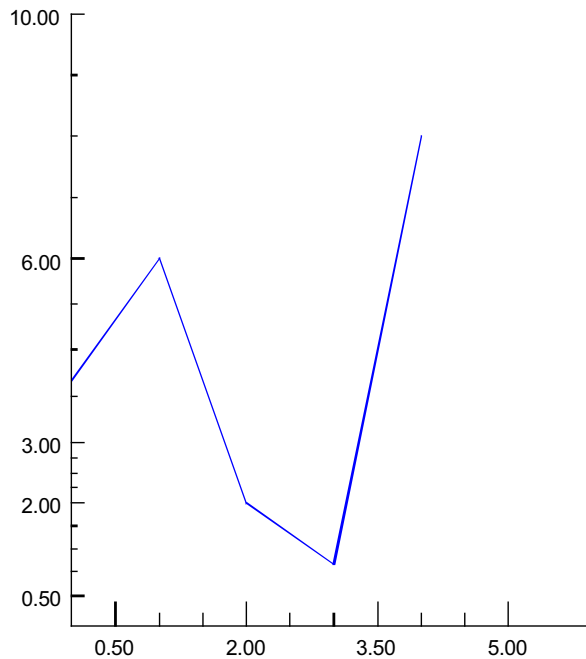
The Ticks attribute, in an Axis1D node, contains the position of the major tick marks. If this attribute is not explicitly set, its value is computed from the attributes FirstTick, TickInterval, Number, Window and Transform.

## Example

This example shows the effects of the tick mark attributes. Note that autoscaling is turned off so that the attribute values are not overridden (see [“Autoscale” on page 89](#)).

On the x-axis, there are four major tick marks (Number), starting with 0.5 (First-Tick) at an interval of 1.5 (TickInterval). There are three minor tick intervals (Density) between each major tick mark. The tick marks are twice as long and are drawn in the opposite direction as normal (TickLength).

On the y-axis, the tick mark locations are set explicitly by the attribute Ticks. This automatically sets the attribute Number. The TickLength is set to -1, so the tick marks are drawn inward (to the right) instead of outward (to the left).



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleTicks extends JFrameChart {

    public SampleTicks() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
```

```

axis.setAutoscaleOutput(axis.AUTOSCALE_OFF);

axis.getAxisX().setWindow(0.0, 6.0);
axis.getAxisX().setDensity(3);
axis.getAxisX().setNumber(4);
axis.getAxisX().setFirstTick(0.5);
axis.getAxisX().setTickInterval(1.5);
axis.getAxisX().setTickLength(-2);

axis.getAxisY().setWindow(0.0, 10.0);
double ticksY[] = {0.5, 2.0, 3.0, 6.0, 10.0};
axis.getAxisY().setTicks(ticksY);
axis.getAxisY().setTickLength(-1);

double y[] = {4, 6, 2, 1, 8};
Data data = new Data(axis, y);
data.setDataType(Data.DATA_TYPE_LINE);
data.setLineColor(Color.blue);
}

public static void main(String argv[]) {
    new SampleTicks().setVisible(true);
}
}

```

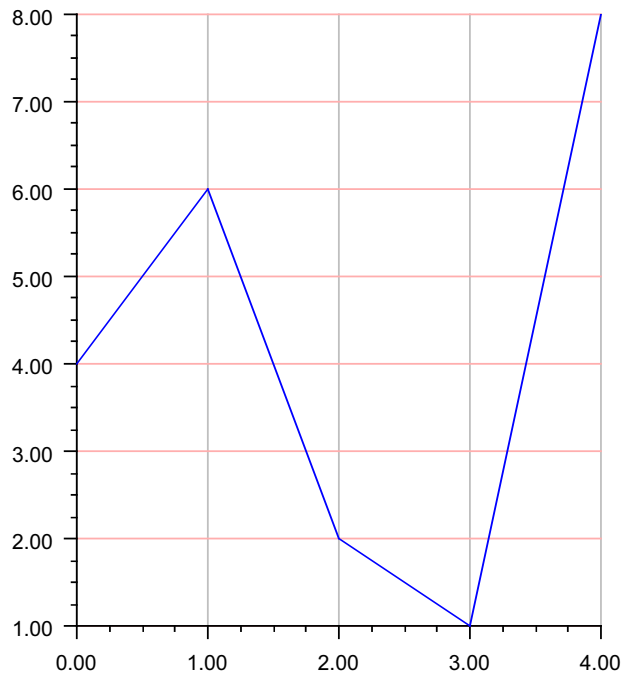
## Grid

The Grid node controls the drawing of grid lines on a chart. The Grid is created by Axis1D as its child. It can be retrieved using the method `Axis1D.getGrid()`.

By default, Grid nodes are not drawn. To enable them, set their "Paint" attribute to `true`. Grid nodes control the drawing of the grid lines perpendicular to their parent axis. So the *x*-axis Grid node controls the drawing of the vertical grid lines.

## Example

In this example, the *x*-axis grid lines are painted light gray and the *y*-axis grid lines are pink.



```
import com.imsl.chart.*;

public class SampleGrid extends JFrameChart {

    public SampleGrid() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        axis.getAxisX().getGrid().setPaint(true);
    }
}
```

```
axis.getAxisY().getGrid().setPaint(true);
axis.getAxisX().getGrid().setLineColor("lightgray");
axis.getAxisY().getGrid().setLineColor("pink");

double y[] = {4, 6, 2, 1, 8};
Data data = new Data(axis, y);
data.setDataType(Data.DATA_TYPE_LINE);
data.setLineColor("blue");
}

public static void main(String argv[]) {
    new SampleGrid().setVisible(true);
}
}
```

## Custom Transform

A custom transformation allows a user-defined mapping of data to an axis. A custom transform is used when the Transform attribute has the value `TRANSFORM_CUSTOM`. The custom transform is the value of the `CustomTransform` attribute.

A custom transform implements the Transform interface. This interface has three methods. One, `setupMapping`, is called first to allow the transformation to initialize itself. The other two, `mapUserToUnit` and `mapUnitToUser`, specify a mapping from the window interval onto  $[0, 1]$ . These methods must each be the inverse of the other.

### Example: Normal Probability Plot

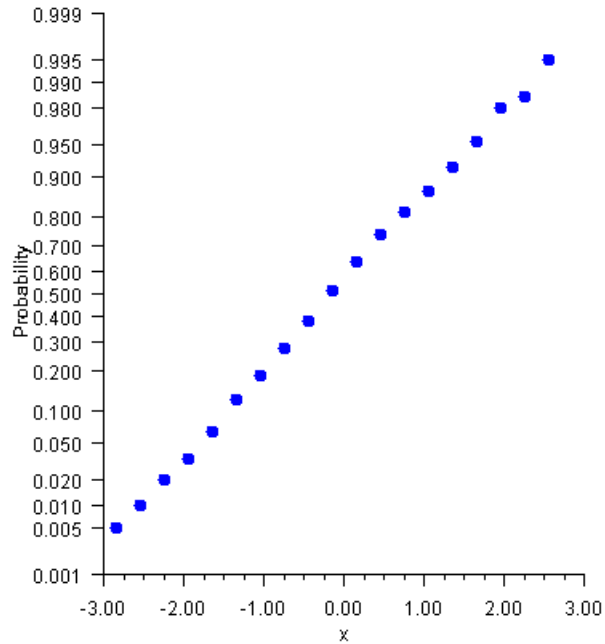
A normal probability plot maps normally distributed data into a straight line, just as a semilog plot maps exponential data into a straight line.

In this example, 400 normally distributed random numbers are grouped into 20 bins. The bin counts are normalized by dividing by the number of samples (400). Cumulative probabilities are computed by summing the probabilities in all of the bins to the left of the current bin.

The custom transformation is the normal cumulative distribution function, `Cdf.normal`, and its inverse, `Cdf.inverseNormal`, scaled to map the probability range  $[0.001, 0.999]$  onto  $[0, 1]$ .

Autoscaling is turned off on the probability ( $y$ ) axis and a fixed set of probability tick marks are specified.

The plot is of the bin center on the  $x$ -axis versus the cumulative probabilities on the  $y$ -axis. The points are not in an exactly straight line because with only a finite number of samples, the distribution does not exactly match the normal distribution.



```

import com.imsl.chart.*;
import com.imsl.stat.Cdf;
import com.imsl.stat.Random;
import java.awt.Color;

public class SampleProbability extends JFrameChart {

    public SampleProbability() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double ticks[] = {0.001, 0.005, 0.01, 0.02, 0.05, 0.10,
            0.20, 0.30,
            0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 0.98,
            0.99, 0.995, 0.999};
        double a = ticks[0];
        double b = ticks[ticks.length-1];

        axis.getAxisX().getAxisTitle().setTitle("x");

        axis.getAxisY().getAxisTitle().setTitle("Probability");
    }
}

```



```

        axis.getAxisY().setTransform(axis.TRANSFORM_CUSTOM);
        axis.getAxisY().setCustomTransform(new
NormalTransform());
        axis.getAxisY().setAutoscaleInput(axis.AUTOSCALE_OFF);
        axis.getAxisY().setWindow(a, b);
        axis.getAxisY().setTextFormat(new
java.text.DecimalFormat("0.000"));
        axis.getAxisY().setTicks(ticks);
        axis.getAxisY().getMinorTick().setPaint(false);

        int nSamples = 400;
        int nBins = 20;

        // Setup the bins
        double bins[] = new double[nBins];
        double dx = 6.0/nBins;
        double x[] = new double[nBins];
        for (int k = 0; k < nBins; k++) {
            x[k] = -3.0 + (k+0.5)*dx;
        }

        // Generate random normal deviates and sort into bins
        Random r = new Random(123457);
        for (int k = 0; k < nSamples; k++) {
            double t = r.nextNormal();
            int j = (int)Math.round((t+3.0-0.5*dx)/dx);
            if (j <= 0) {
                bins[0]++;
            } else if (j >= nBins-1) {
                bins[nBins-1]++;
            } else {
                bins[j]++;
            }
        }

        // Compute the cumulative distribution
        // y[k] is the sum of bins[j] for j=0,...,k divided
by the nSamples.
        double y[] = new double[nBins];
        y[0] = bins[0]/nSamples;
        for (int k = 1; k < nBins; k++) {
            y[k] = y[k-1] + bins[k]/nSamples;
        }

```

```

        // Plot the data using markers
        Data data = new Data(axis, x, y);
        data.setDataType(Data.DATA_TYPE_MARKER);
        data.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
        data.setMarkerColor(Color.blue);
    }

    public static void main(String argv[]) {
        new SampleProbability().setVisible(true);
    }

    static class NormalTransform implements Transform {
        private double scaleA, scaleB;

        /**
         * Initializes the mappings between user and coordi-
         nate space.
         */
        public void setupMapping(Axis1D axis1d) {
            double w[] = axis1d.getWindow();
            double t = Cdf.inverseNormal(w[0]);
            scaleB = 1.0/(Cdf.inverseNormal(w[1])-t);
            scaleA = -scaleB*t;
        }

        /**
         * Maps a point in [0,1] to a probability.
         */
        public double mapUnitToUser(double unit) {
            return Cdf.normal((unit-scaleA)/scaleB);
        }

        /**
         * Maps a probability to the interval [0,1].
         */
        public double mapUserToUnit(double p) {
            return scaleA + scaleB*Cdf.inverseNormal(p);
        }
    }
}

```

## Multiple Axes

A JMSL chart can contain any number of axes. Each axis has its own mapping from the user coordinates to the device (screen) coordinates.

Normally, the  $x$ -axis is at the bottom of the chart and the  $y$ -axis is to the left. The attribute `Type` can be changed to move the  $x$ -axis to the top of the chart and/or the  $y$ -axis to the right of the chart.

Axis can be moved from the chart edge, either away from the chart or into the middle of the chart data area, by setting the attribute `Cross`.

### Attribute Type

The attribute `Type` specifies the position of an  $x$  or  $y$ -axis. Applied to the  $x$ -axis it can have the values `AXIS_X` (the default) or `AXIS_X_TOP`. Applied to the  $y$ -axis, it can have the value `AXIS_Y` (the default) or `AXIS_Y_RIGHT`.

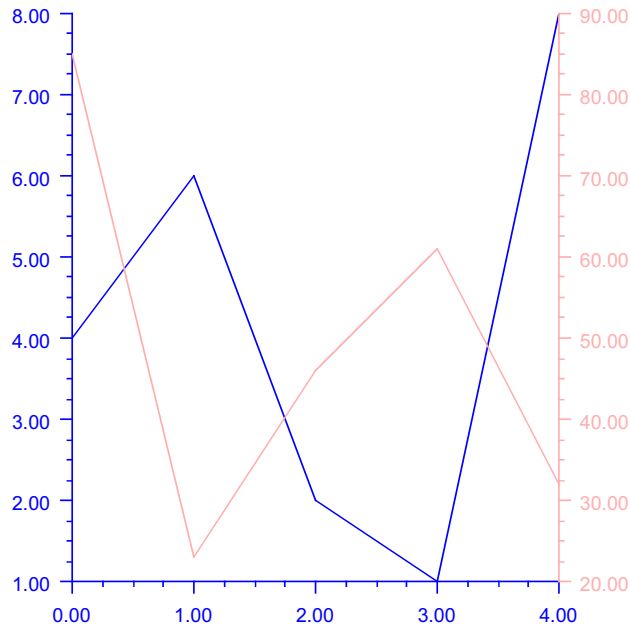
### Attribute Cross

The `Cross` attribute specifies the coordinates of intersection of the  $x$ -axis and  $y$ -axis. This can be inside or outside of the chart body. `Cross` can be used to place multiple  $y$ -axes to the left of the chart or multiple  $x$ -axes below the chart.

## Example

Two data sets are plotted, each on its own axis. The first (blue) axis is left in the default position with the y-axis on the left. The second (pink) axis has its y-axis moved to the left.

In this example the x-axis is shared between the two axes, so it is not painted as part of the second (pink) axis. This is done by setting its Paint attribute to false.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleAxesLeftRight extends JFrameChart {

    public SampleAxesLeftRight() {
        Chart chart = getChart();

        AxisXY axisLeft = new AxisXY(chart);
        double yLeft[] = {4, 6, 2, 1, 8};
        Data dataLeft = new Data(axisLeft, yLeft);
        dataLeft.setDataType(Data.DATA_TYPE_LINE);
        axisLeft.setLineColor(Color.blue);
        axisLeft.setTextColor(Color.blue);
    }
}
```

```
AxisXY axisRight = new AxisXY(chart);
axisRight.getAxisX().setPaint(false);
axisRight.getAxisY().setType(Axis1D.AXIS_Y_RIGHT);
double yRight[] = {85, 23, 46, 61, 32};
Data dataRight = new Data(axisRight, yRight);
dataRight.setDataType(Data.DATA_TYPE_LINE);
axisRight.setLineColor(Color.pink);
axisRight.setTextColor(Color.pink);
}

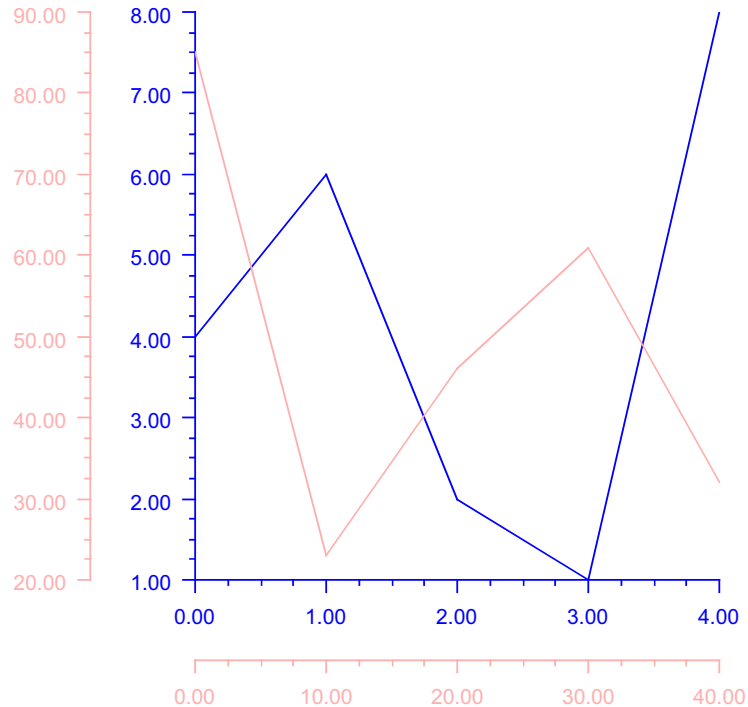
public static void main(String argv[]) {
    new SampleAxesLeftRight().setVisible(true);
}
}
```

## Cross Example

Multiple  $x$  and  $y$ -axes are shown.

The Cross attribute is used to specify that the second (pink) axes intersect at (-8,10), in the pink-axis coordinate system.

The Viewport attribute is changed in both sets of axes to shrink the size of the chart body and leave more room to the left and below the chart for the pink axes.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleAxesCross extends JFrameChart {

    public SampleAxesCross() {
        Chart chart = getChart();

        AxisXY axisLeft = new AxisXY(chart);
        double yLeft[] = {4, 6, 2, 1, 8};
        Data dataLeft = new Data(axisLeft, yLeft);
        dataLeft.setDataType(Data.DATA_TYPE_LINE);
    }
}
```

```

axisLeft.setLineColor(Color.blue);
axisLeft.setTextColor(Color.blue);

AxisXY axisRight = new AxisXY(chart);
axisRight.setCross(-8,10);
double xRight[] = {0, 10, 20, 30, 40};
double yRight[] = {85, 23, 46, 61, 32};
Data dataRight = new Data(axisRight, xRight, yRight);
dataRight.setDataType(Data.DATA_TYPE_LINE);
axisRight.setLineColor(Color.pink);
axisRight.setTextColor(Color.pink);

double viewport[] = {0.3, 0.9, 0.05, 0.7};
axisLeft.setViewport(viewport);
axisRight.setViewport(viewport);
}

public static void main(String argv[]) {
    new SampleAxesCross().setVisible(true);
}
}

```

---

## Background

Background controls the drawing of the chart's background. It is created by Chart as its child. It can be retrieved from a Chart object using the `Chart.getBackground()` method.

The fill area attributes in the Background node determine how the background is drawn (see [“Fill Area Attributes” on page 75](#)).

The attribute `FillType` has the global default value of `FILL_TYPE_SOLID`. The attribute `FillColor` attribute is set to `Color.white` in this node.

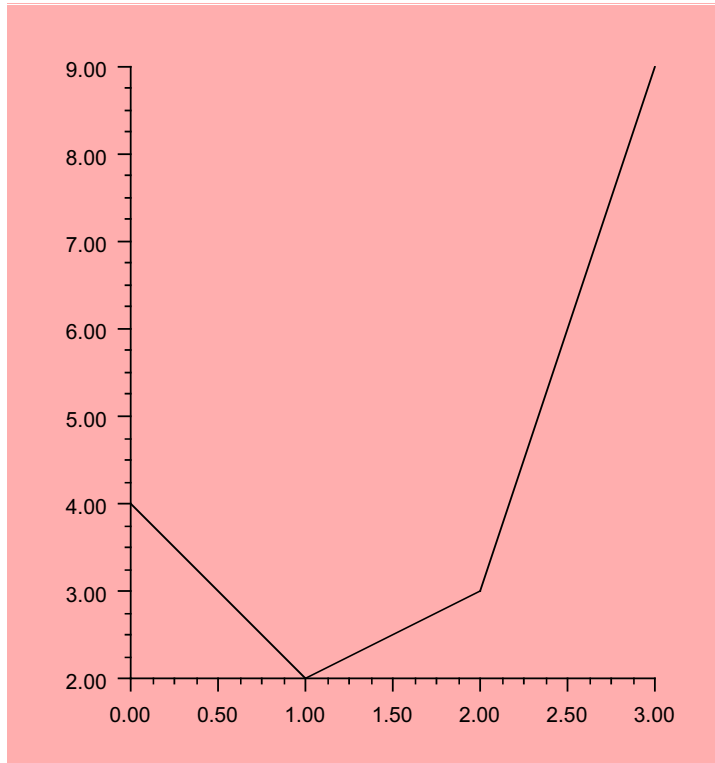
### Solid Color Background

To set the background to a solid color:

- set the attribute `FillType` to `FILL_TYPE_SOLID`, and
- set the attribute `FillColor` to the desired color.

For example the following code sets the background to pink. To view chart in color please see the online documentation.





```
import com.imsl.chart.*;

public class SampleBackgroundSolid extends JFrameChart {

    public SampleBackgroundSolid() {
        Chart chart = getChart();
        chart.getBackground().setFillType(ChartNode.FILL_TYPE_SOLID);
        chart.getBackground().setFillColor(java.awt.Color.pink);
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);
    }

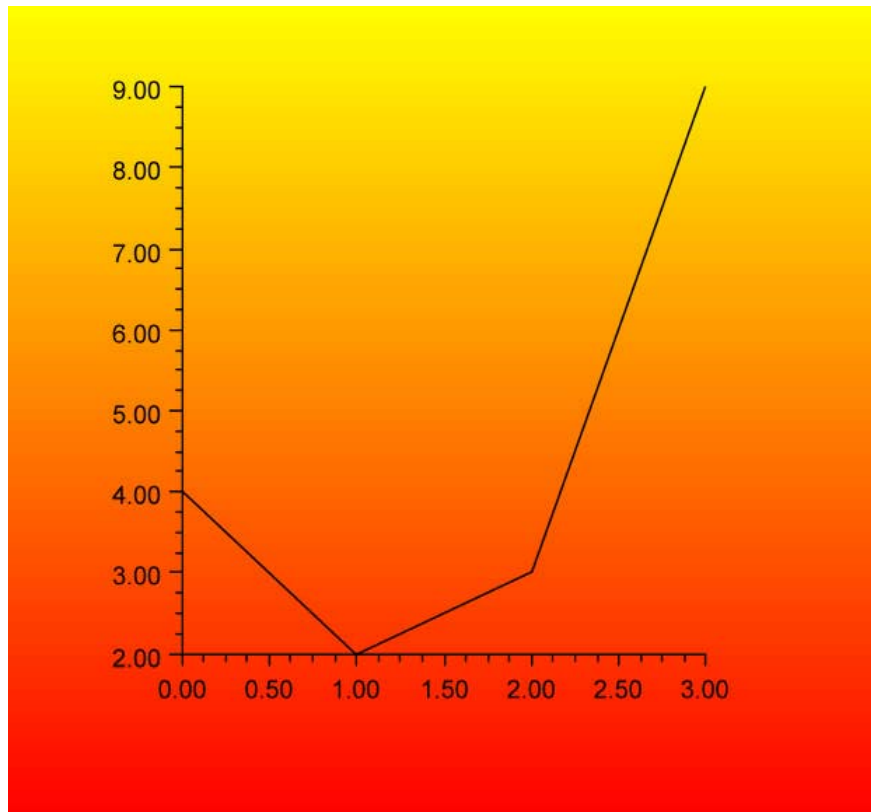
    public static void main(String argv[]) {
        new SampleBackgroundSolid().setVisible(true);
    }
}
```

## Gradient Color Background

To set the background to a color gradient:

- set the attribute `FillType` to `FILL_TYPE_GRADIENT`, and
- set the attribute `Gradient` to the desired color gradient specification.

For example the following code uses a yellow-to-red vertical gradient for the background setting. See [“Fill Area Attributes” on page 75](#) for more information on gradients.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleBackgroundGradient extends JFrameChart {

    public SampleBackgroundGradient() {
        Chart chart = getChart();
        chart.getBackground().setFillType(ChartNode.FILL_TYPE_GRADIENT);
    }
}
```

```
        chart.setBackground().setGradient(Color.yellow, Color.yellow,
Color.red, Color.red);
        AxisXY axis = new AxisXY(chart);
        double y[] = {4, 2, 3, 9};
        new Data(axis, y);
    }

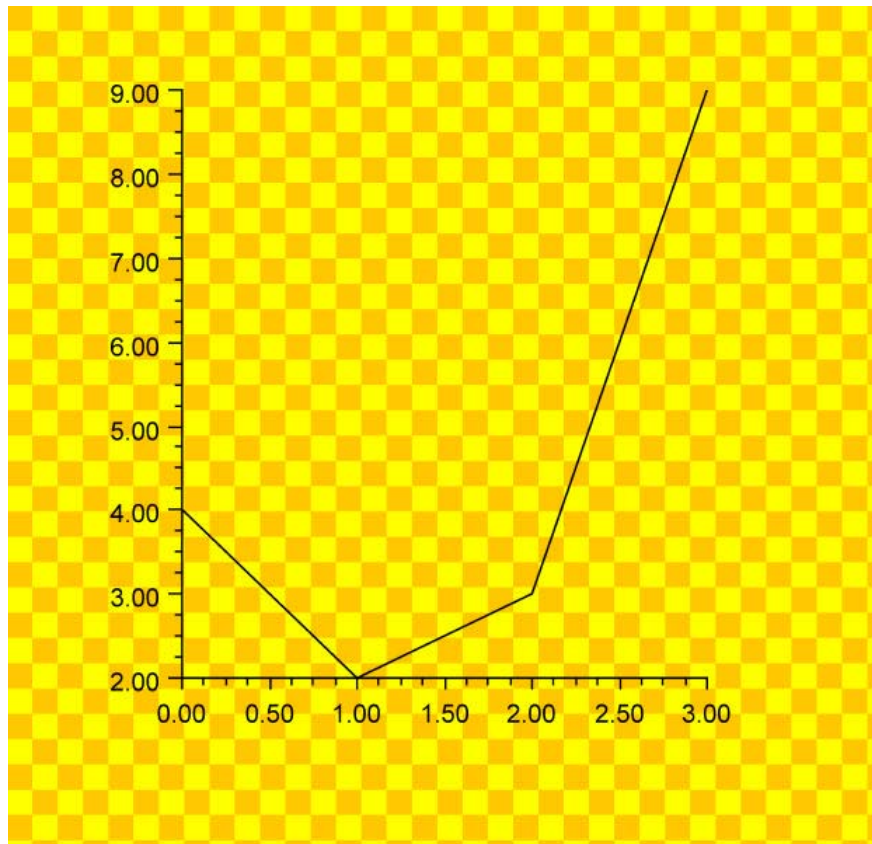
    public static void main(String argv[]) {
        new SampleBackgroundGradient().setVisible(true);
    }
}
```

## Pattern Background

To set the background to a color pattern:

- set the attribute `FillType` to `FILL_TYPE_PAINT`, and
- set the attribute `FillPaint` to the desired pattern.

For example the following code sets the background to yellow/orange checkerboard pattern. See [“Fill Area Attributes” on page 75](#) for more information on patterns.



```
import com.imsl.chart.*;
import java.awt.Color;
import java.awt.Paint;

public class SampleBackgroundPaint extends JFrameChart {

    public SampleBackgroundPaint() {
```

```
    Chart chart = getChart();
    chart.getBackground().setFillType(ChartNode.FILL_TYPE_PAINT);
    Paint paint = FillPaint.checkerboard(24, Color.yellow,Color.orange);
    chart.getBackground().setFillPaint(paint);
    AxisXY axis = new AxisXY(chart);
    double y[] = {4, 2, 3, 9};
    new Data(axis, y);
}

public static void main(String argv[]) {
    new SampleBackgroundPaint().setVisible(true);
}
}
```

---

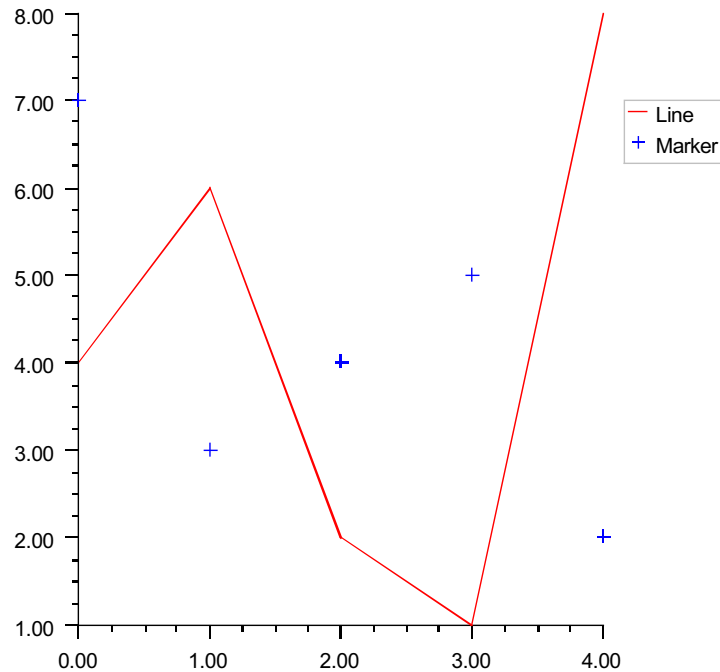
## Legend

The legend is used to identify data sets. Data nodes that have their Title attribute defined are automatically included in the legend box.

The Legend node is automatically created by the Chart node as its child. By default, the legend is not drawn because its Paint attribute is set to false.

### Simple Legend Example

At a minimum, adding a legend to a chart requires setting the legend's Paint attribute to true and setting the Title attribute in the Data nodes that are to appear in the legend box. This example shows such a minimal legend.



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleLegend extends JFrameChart {

    public SampleLegend() {
```

```

Chart chart = getChart();
AxisXY axis = new AxisXY(chart);

Legend legend = chart.getLegend();
legend.setPaint(true);
legend.setTitle("Legend");
legend.setTextColor(Color.white);
legend.setFillType(legend.FILL_TYPE_SOLID);
legend.setViewport(0.3, 0.4, 0.1, 0.4);

double y1[] = {4, 6, 2, 1, 8};
Data data1 = new Data(axis, y1);
data1.setDataType(Data.DATA_TYPE_LINE);
data1.setLineColor(Color.red);
data1.setTitle("Line");

double y2[] = {7, 3, 4, 5, 2};
Data data2 = new Data(axis, y2);
data2.setDataType(Data.DATA_TYPE_MARKER);
data2.setMarkerColor(Color.blue);
data2.setTitle("Marker");
}

public static void main(String argv[]) {
    new SampleLegend().setVisible(true);
}
}

```

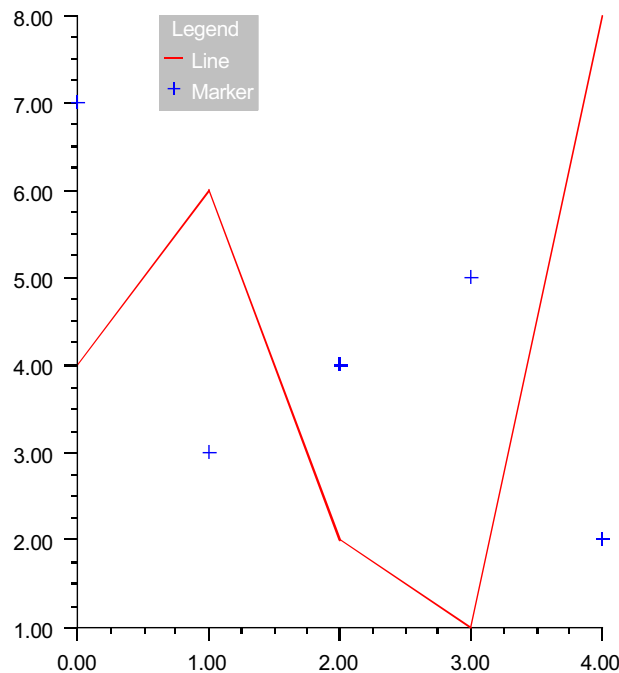
## Legend Example

This example shows more of the attributes that affect a legend. If the legend's Title attribute is set, then it is used as a header in the legend box.

The text properties for all of the text strings in the legend box are obtained from the Legend node, not from the associated Data nodes (see [“Text Attributes” on page 79](#)). Here the TextColor is set to white.

The background of the legend box can be set by changing the fill attributes (see [“Fill Area Attributes” on page 75](#)). By default in the Legend node, FillType is set to FILL\_TYPE\_NONE and FillColor is set to Color.lightGray.

The position of the legend box is controlled by its Viewport attribute. The viewport is the region of the Component, in which the chart is being drawn, that the legend box occupies. The upper left corner is (0,0) and the lower right corner is (1,1). The default value of the legend viewport is [0.83, 0.0] by [0.2, 0.2].



```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleSimpleLegend extends JFrameChart {

    public SampleSimpleLegend() {
```



```

Chart chart = getChart();
AxisXY axis = new AxisXY(chart);
chart.getLegend().setPaint(true);

double y1[] = {4, 6, 2, 1, 8};
Data data1 = new Data(axis, y1);
data1.setDataTypes(Data.DATA_TYPE_LINE);
data1.setLineColor(Color.red);
data1.setTitle("Line");

double y2[] = {7, 3, 4, 5, 2};
Data data2 = new Data(axis, y2);
data2.setDataTypes(Data.DATA_TYPE_MARKER);
data2.setMarkerColor(Color.blue);
data2.setTitle("Marker");
}

public static void main(String argv[]) {
    new SampleSimpleLegend().setVisible(true);
}
}

```

---

## Tool Tips

Swing supports tool tips, small help boxes that pop up when the mouse hovers over some element. The `ToolTip` node allows tool tips to be associated with chart elements. The text displayed in the tool tip is the `Title` attribute of the `ToolTip`'s parent node.

### Example

In this example two `Data` nodes are created. Each `Data` node has a `Title` defined and a `ToolTip` node added as a child. Note that the `ToolTip` node just has to be created, no methods using it are normally required.

```
import com.imsl.chart.*;
import java.awt.Color;

public class SampleToolTip extends JFrameChart {

    public SampleToolTip() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);

        double y1[] = {4, 6, 2, 1, 8};
        Data data1 = new Data(axis, y1);
        data1.setDataType(Data.DATA_TYPE_MARKER);
        data1.setMarkerColor(Color.red);
        data1.setMarkerType(Data.MARKER_TYPE_FILLED_CIRCLE);
        data1.setTitle("Set A");
        new ToolTip(data1);

        double y2[] = {7, 3, 4, 5, 2};
        Data data2 = new Data(axis, y2);
        data2.setDataType(Data.DATA_TYPE_MARKER);
        data2.setMarkerColor(Color.blue);
        data2.setMarkerType(Data.MARKER_TYPE_FILLED_SQUARE);
        data2.setTitle("Set B");
        new ToolTip(data2);
    }

    public static void main(String argv[]) {
        new SampleToolTip().setVisible(true);
    }
}
```



# Chapter 4: XML

A JMSL chart tree can be created from an XML file. XML (Extensible Markup Language) is a universal format for structured data. XML looks similar to HTML but has application-dependent tag names instead of HTML tags (<p>, <h1>, etc.). JMSL defines a set of tags for describing charts.

The JMSL distribution includes the “Crimson 1.1.3” XML parser from Apache at `lib/crimson.jar`. *The XML features of JMSL require that either JDK 1.4 or later, crimson.jar, or another XML parser that supports the JAXP API be used.*

---

## Chart XML Syntax

The Chart XML syntax is defined by a document type definition (DTD) file at <http://www.vni.com/products/jmsl/Chart.dtd>.

Valid XML files begin with a prolog that contains the XML declaration and the document type declaration. For example,

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//
EN" "http://www.vni.com/products/jmsl/Chart.dtd">
```

Following the prolog is the body of the chart XML file. This is a tree of tags, with the <Chart> tag at the root. These tags mirror a JMSL chart tree. The following outline shows the hierarchy of tags.

```
<Chart >
  <Array> | <Array id=name>
  <Background>
  <ChartTitle>
  <Legend>
  <AxisXY>
    <AxisX> | <AxisY>
      <AxisTitle>
      <AxisLabel>
      <AxisLine>
      <AxisUnit>
      <Grid>
      <MajorTick>
      <MinorTick>
  <Data x=array y=array> | <Data y=array> | <Data>
  <Bar x=array y=array> | <Bar y=array>
    <BarSet> | <BarSet index=int>
      <BarItem> | <BarItem index=int>
        <ErrorBar x=array y=array low=array high=array>
        <HighLowClose start=array high=array low=array
close=array>
        <HighLowClose high=array low=array close=array>
  <Polar>
  <Pie y=array>
  <PieSlice> | <PieSlice index=int>
```

Some of the tags have array-valued arguments. These are either a list of doubles of the form "{1.2,3.4,5}" or "#id", where *id* is an <Array> tag id.

Each of the chart tags can also have any number of <Attribute> tags as children. They can have either of two forms:

```
<Attribute name="attribute-name" value="attribute-value"/>
```

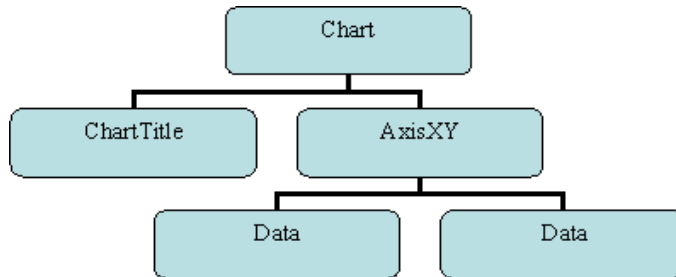
**or**

```
<Attribute name="attribute-name">attribute-value</Attribute>
```

Also, while not shown in the above list, each chart node can have a <ToolTip> node as a child.

## Syntax Example

The chart tree



is created from the following XML file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <ChartTitle>
    <Attribute name="Title">Line Chart</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <AxisXY>
    <Data y="{6,5,7,1}">
      <Attribute name="LineColor" value="blue"/>
      <Attribute name="LineWidth" value="2.0"/>
      <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE"/>
      <Attribute name="Title" value="Blue Line"/>
    </Data>
    <Data y="{1,3,6,8}">
      <Attribute name="MarkerColor" value="darkblue"/>
      <Attribute name="DataType" value="DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE"/>
      <Attribute name="Title" value="Markers"/>
    </Data>
  </AxisXY>
</Chart>
```

In the ChartTitle node there are three attributes defined: Title, FontSize and Font-Style. The argument "y={6,5,7,1}" in the Data node is used to create an array passed to the Data constructor.

Note that the attribute name must be specified using "name=", but that the attribute's value can be either the value of a "value=" or the body of the attribute tag.

---

## Attribute Tags

Attribute tags can have either of two forms:

```
<Attribute name="attribute-name" value="attribute-value"/>
```

**or**

```
<Attribute name="attribute-name">attribute-value</Attribute>
```

The type of an attribute's value is determined by the return type of its getter method. For example, if the attribute type is Foo, its type is the return type of the 0-argument method getFoo() defined in the node being defined or in one of its ancestor nodes.

Symbolic names can be used for enumerated types. See [“Enumerated Types in Chart XML Files” on page 137](#) for details.

Attribute values of the following types can be handled: String, Color, int or Integer, double or Double, Double[], boolean or Boolean, Image, Paint, or Locale.

Colors are either red, green, blue triples in the range [0,255] or color names. For example "255,215,0" or "gold".

A locale can be either the name of one of the static, public fields in Locale, such as ENGLISH or of the form language\_country or language\_country\_variant. For example, "nl\_BE" or "nl\_BE\_EURO".

Attribute values of type Image and Paint are specified by giving the URL of an image file, for example

```
"http://www.google.com/images/logo.gif"
```

As a special case, the URL protocol can be a classpath, for example

```
"classpath:/com/ims1/example/chart/marker.gif"
```

The image is then loaded as a resource by the classloader using getResource(String). The classloader used is the one used to load the ChartXML class. This feature is used to load images from JAR files.

---

## Array Tags

The array tag is used to define data. The `id` argument is used to name the array. Array tags can be nested to form multidimensional arrays. At the inner most level the following tags are allowed.

- `<String value=string>`
- `<Color value=color>`

The value of `color` can be either a red, green, blue triple in the range `[0,255]` or color names. For example `"255, 215, 0"` or `"gold"`.

- `<Date value=date style=style locale=locale>`

If the argument `locale` is specified, it is parsed the same as the `locale` argument in the tag `Attribute` as described above. If the `locale` is not specified, then the default `locale` is used.

If the argument `style` is `SHORT`, `MEDIUM`, `LONG` or `FULL`, then the date is parsed using the object `DateFormat.getDateInstance(style,locale)`. If `style` is not specified, then `SHORT` is assumed.

`Style` can also be a pattern used to construct a `SimpleDateObject(style,locale)`.

- `<Number value=number>`

Numbers are parsed using `Double.valueOf(number)`.

- `<NumberList>number,...,number</NumberList> | <NumberList value=list>`

The body of this tag is a comma separated list of numbers. Each number is parsed using `Double.valueOf(number)`.

The value arguments are optional. If omitted, the tag's body is used instead.



## Array Tag Examples

The first example is of a three-dimensional array called `foo`.

```
<Array id="foo">
  <Array>
    <Array><NumberList value="1,2,3"/></Array>
    <Array><NumberList value="4,5,6"/></Array>
  </Array>
  <Array>
    <Array><NumberList value="11,12,13"/></Array>
    <Array><NumberList value="14,15,16"/></Array>
  </Array>
</Array>
```

This corresponds to the Java declaration

```
double foo[][][] = {{{1,2,3},{4,5,6}},{11,12,13},{14,15,16}}}
```

The second example is an array of three strings called `states`.

```
<Array id="states">
  <String value="NJ" />
  <String value="TX" />
  <String value="CO" />
</Array>
```

The final example is an array of four colors, called `bg`.

```
<Array id="bg">
  <Color value="yellow" />
  <Color value="0,128,64" />
  <Color value="lightblue" />
  <Color value="yellow" />
</Array>
```

---

## Creating a Chart from XML

The class `ChartXML` creates a chart tree from an XML file. It can be used directly as an application to view an XML file. The following command displays the chart described by the XML file *filename*. (This assumes that the `CLASSPATH` is correctly set up.)

```
java com.imsi.chart.xml.ChartXML filename
```

## Interacting with an XML Created Chart

The class `ChartXML` can also be used to create a chart tree within a larger Java program. The object's `getChart()` method is used to retrieve the created chart object. The `get(String)` method is used to obtain objects created by array tags.

---

## Enumerated Types in Chart XML Files

In a chart XML file, the following symbolic names can be used for enumerated values. Their numerical equivalents can also be used.

These enumerated types can be "or-ed" together. For example,

```
<Attribute name="DataType"
  value=DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
```

is allowed.

### Marker Types

```
MARKER_TYPE_PLUS
MARKER_TYPE_ASTERISK
MARKER_TYPE_X
MARKER_TYPE_HOLLOW_SQUARE
MARKER_TYPE_FILLED_SQUARE
MARKER_TYPE_HOLLOW_TRIANGLE
MARKER_TYPE_FILLED_TRIANGLE
MARKER_TYPE_HOLLOW_DIAMOND
MARKER_TYPE_FILLED_DIAMOND
MARKER_TYPE_DIAMOND_PLUS
MARKER_TYPE_SQUARE_X
MARKER_TYPE_SQUARE_PLUS
MARKER_TYPE_OCTAGON_X
MARKER_TYPE_OCTAGON_PLUS
MARKER_TYPE_HOLLOW_CIRCLE
MARKER_TYPE_FILLED_CIRCLE
MARKER_TYPE_CIRCLE_X
```

MARKER\_TYPE\_CIRCLE\_PLUS  
MARKER\_TYPE\_CIRCLE\_CIRCLE

## Fill Types

FILL\_TYPE\_NONE  
FILL\_TYPE\_SOLID  
FILL\_TYPE\_GRADIENT  
FILL\_TYPE\_PAINT

## Text Alignments

TEXT\_X\_LEFT  
TEXT\_X\_CENTER  
TEXT\_X\_RIGHT  
TEXT\_Y\_BOTTOM  
TEXT\_Y\_CENTER  
TEXT\_Y\_TOP

## Font Styles

PLAIN  
BOLD  
ITALIC

## Axis Parameters

AXIS\_X  
AXIS\_X\_TOP  
AXIS\_Y  
AXIS\_Y\_RIGHT

## **Autoscale Parameters**

AUTOSCALE\_OFF  
AUTOSCALE\_DATA  
AUTOSCALE\_WINDOW  
AUTOSCALE\_NUMBER

## **Axis Transforms**

TRANSFORM\_LINEAR  
TRANSFORM\_LOG

## **Data Node Types**

DATA\_TYPE\_LINE  
DATA\_TYPE\_MARKER  
DATA\_TYPE\_FILL  
DATA\_TYPE\_PICTURE

## **Label Types**

LABEL\_TYPE\_NONE  
LABEL\_TYPE\_TITLE  
LABEL\_TYPE\_Y  
LABEL\_TYPE\_X  
LABEL\_TYPE\_PERCENT

## **Bar Types**

BAR\_TYPE\_VERTICAL  
BAR\_TYPE\_HORIZONTAL

---

## Creating Charts from General XML Files Using XSLT

JMSL can create charts from XML files that are written using a particular set of tags, but you can also generate charts from more general XML files. This is done by using XSL Transformations (XSLT) to transform other types of XML files into the type of XML file required by JMSL.

### Example

Consider the following sales report in XML that does not use the tag names and structure that JMSL requires.

```
<?xml version="1.0" encoding="UTF-8" ?>
<sales>
  <region>
    <name>North America</name>
    <amount>102.890</amount>
  </region>
  <region>
    <name>Europe</name>
    <amount>87.654</amount>
  </region>
  <region>
    <name>Japan</name>
    <amount>93.895</amount>
  </region>
  <region>
    <name>Rest of World</name>
    <amount>37.985</amount>
  </region>
</sales>
```

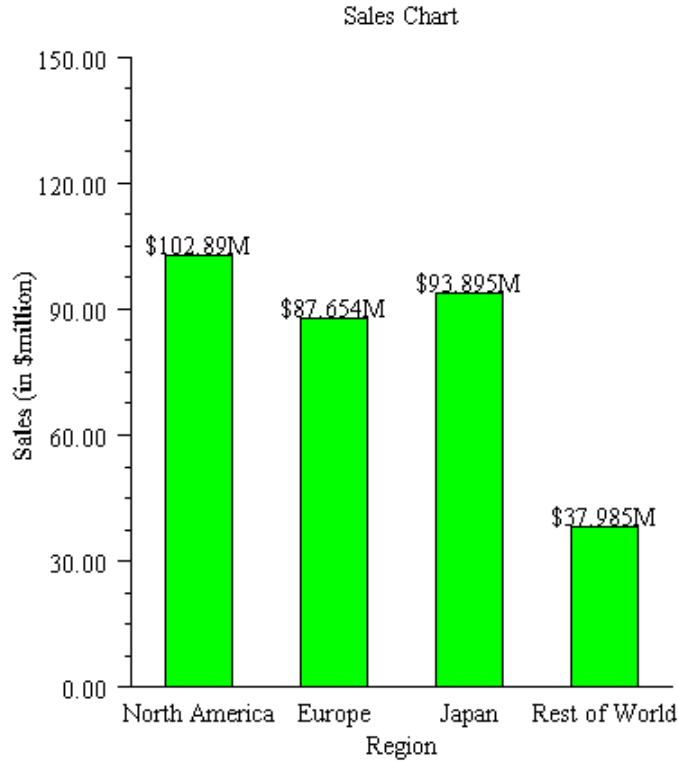
To create a bar chart from this XML, it must be transformed into an XML file similar to the following

```
<?xml version="1.0" encoding="UTF-8" ?>
<Chart>
  <Array id="amount">
    <Number>102.890</Number>
    <Number>87.654</Number>
    <Number>93.895</Number>
    <Number>37.985</Number>
  </Array>
```

```

<Array id="region">
  <String value="North America"/>
  <String value="Europe"/>
  <String value="Japan"/>
  <String value="Rest of World"/>
</Array>
<Attribute name="FontName" value="Serif"/>
<Attribute name="FontSize" value="14"/>
<ChartTitle><Attribute name="Title" value="Sales Chart"/></ChartTitle>
<AxisXY>
  <AxisY>
    <AxisTitle><Attribute name="Title" value="Sales (in $million)"/>
    </AxisTitle>
  </AxisY>
  <AxisX>
    <AxisTitle><Attribute name="Title" value="Region"/></AxisTitle>
  </AxisX>
  <Bar y="#amount">
    <Attribute name="BarType" value="BAR_TYPE_VERTICAL"/>
    <Attribute name="Labels" value="#region"/>
    <Attribute name="LabelType" value="LABEL_TYPE_Y"/>
    <Attribute name="FillColor" value="green"/>
    <Attribute name="TextFormat">${###.###}M</Attribute>
  </Bar>
</AxisXY>
</Chart>

```



The following XSLT file can be used to perform this transformation.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
<Chart>
  <Array id="region">
    <xsl:for-each select="sales/region/name">
      <String><xsl:apply-templates/></String>
    </xsl:for-each>
  </Array>
  <Array id="amount">
    <xsl:for-each select="sales/region/amount">
      <Number><xsl:apply-templates/></Number>
    </xsl:for-each>
  </Array>
</Chart>
</template>
</xsl:stylesheet>
```

```

</Array>
<Attribute name="FontName" value="Serif"/>
<Attribute name="FontSize" value="14"/>
<ChartTitle><Attribute name="Title" value="Sales Chart"/></ChartTitle>
<AxisXY>
  <AxisY>
    <AxisTitle>
      <Attribute name="Title" value="Sales (in $million)"/>
    </AxisTitle>
  </AxisY>
  <AxisX>
    <AxisTitle><Attribute name="Title" value="Region"/></AxisTitle>
  </AxisX>
  <Bar y="#amount">
    <Attribute name="BarType" value="BAR_TYPE_VERTICAL"/>
    <Attribute name="Labels" value="#region"/>
    <Attribute name="LabelType" value="LABEL_TYPE_Y"/>
    <Attribute name="FillColor" value="green"/>
    <Attribute name="TextFormat">$$$#.###M</Attribute>
  </Bar>
</AxisXY>
</Chart>
</xsl:template>

</xsl:stylesheet>

```

While a complete discussion of XSLT is beyond the scope of this manual, note that most of the file is the same as the target. The XSL tags are in the "xsl" namespace. In this example, the key XSL tags are the two `for` loops that reformat the data. For instance, the region names are formatted by the fragment

```

<xsl:for-each select="sales/region/name">
  <String><xsl:apply-templates/></String>
</xsl:for-each>

```

where the path "sales/region/name" selects tags in the original XML file and the body of the tag

```

<String><xsl:apply-templates/></String>

```

is copied into the new file.

The given XSL document will work for similar documents with different data, but other XML document formats may require different XSL files.



The Java code required to implement this transformation is:

```
import com.imsl.chart.JFrameChart;
import com.imsl.chart.xml.ChartXML;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.dom.DOMResult;
import org.w3c.dom.Document;

public class SampleSalesTransform {
    public static void main(String argv[]) throws Exception {
        String filenameXSL = argv[0];
        String filenameXML = argv[1];

        // Create an XML parser factory
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setNamespaceAware(true);
        DocumentBuilder builder = factory.newDocumentBuilder();

        // Parse the XSL file as an XML file
        Document docXSL = builder.parse(filenameXSL);
        DOMSource sourceXSL = new DOMSource(docXSL);
        sourceXSL.setSystemId(filenameXSL);

        // Create a transformation based on the XSL file
        TransformerFactory tFactory = TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer(sourceXSL);

        // Parse the input XML file
        Document docXML = builder.parse(filenameXML);
        DOMSource sourceXML = new DOMSource(docXML);
        sourceXML.setSystemId(filenameXML);

        // Transform the input XML file using the XSL transformer
        DOMResult result = new DOMResult();
        transformer.transform(sourceXML, result);

        // Create a chart from the transformed XML
        ChartXML chartXML = new ChartXML((Document)result.getNode());
        new JFrameChart(chartXML.getChart()).show();
    }
}
```

This program takes two arguments, the names of the XSL and XML files, and creates a JMSL chart. The transformed XML exists only in memory and is not written

to disk. This class will work with any (correct) XSL and XML files; it is not dependent on the details of this example. In the interests of simplicity, error handling and validation is not implemented, as they should be for a production implementation.

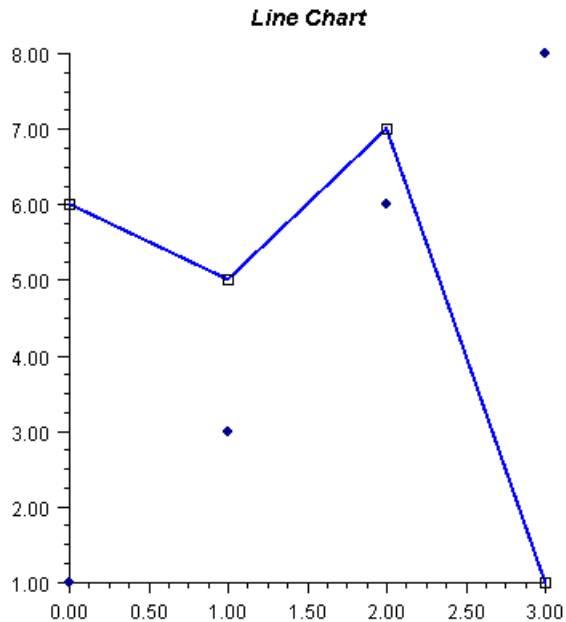
Also, for simplicity, this example is a desktop application. XML and XSLT are generally more useful as part of a server-side Web application. The above code can be used with `com.ims1.chart.JspBean` to create a server-side XML/XSLT solution.

---

## XML Examples

- [Line Chart](#) (see page 146)
- [Axis Titles](#) (see page 148)
- [Multiple Axes](#) (see page 149)
- [Axis Offset](#) (see page 151)
- [Rotated Labels](#) (see page 153)
- [Pie Chart](#) (see page 158)
- [Box Plot](#) (see page 155)
- [Contour Chart](#) (see page 160)
- [Large Bar Chart](#) (see page 162)
- [Grouped and Stacked Bar Chart](#) (see page 165)
- [Polar Plot](#) (see page 167)

### XML Example: Line Chart



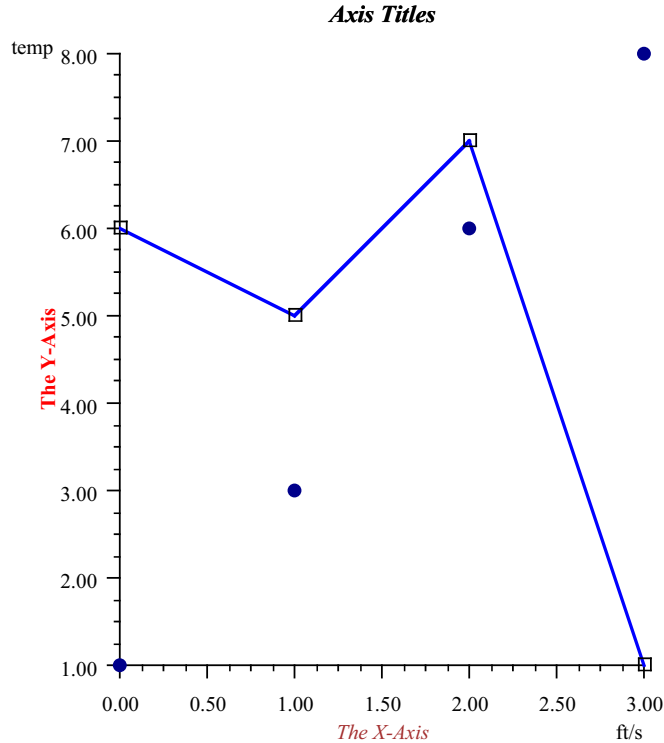
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

```

<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <ChartTitle>
    <Attribute name="Title">Line Chart</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <AxisXY>
    <Data y="{6,5,7,1}">
      <Attribute name="LineColor" value="blue"/>
      <Attribute name="LineWidth" value="2.0"/>
      <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE"/>
      <Attribute name="Title" value="Blue Line"/>
      <ToolTip/>
    </Data>
    <Data y="{1,3,6,8}">
      <Attribute name="MarkerColor" value="darkblue"/>
      <Attribute name="DataType" value="DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE"/>
      <Attribute name="Title" value="Markers"/>
      <ToolTip/>
    </Data>
  </AxisXY>
</Chart>

```

## XML Example: Axis Titles



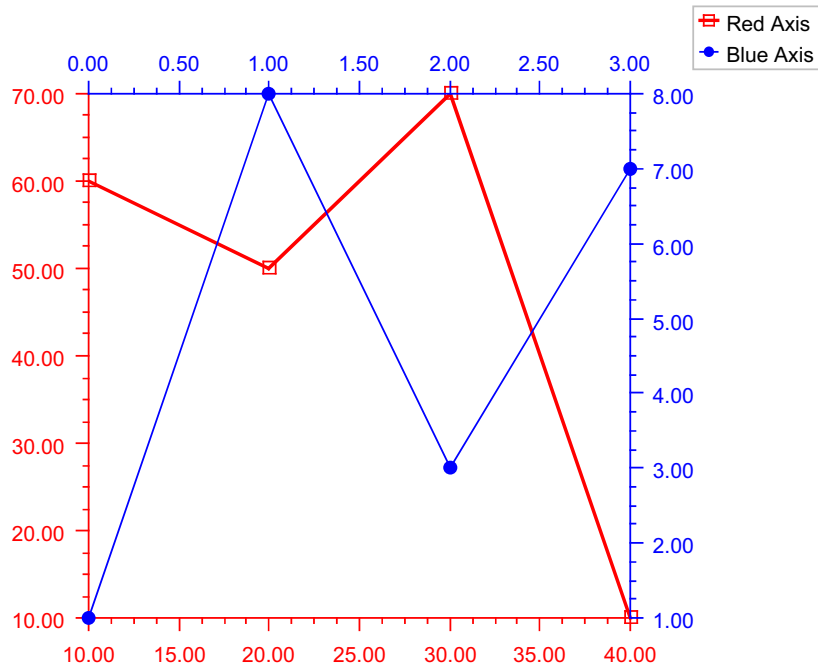
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <Attribute name="FontName" value="Serif"/>
  <ChartTitle>
    <Attribute name="Title">Axis Titles</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <AxisXY>
    <AxisX>
      <AxisTitle>
        <Attribute name="Title" value="The X-Axis"/>
        <Attribute name="TextColor" value="brown"/>
        <Attribute name="FontStyle" value="ITALIC"/>
      </AxisTitle>
      <AxisUnit><Attribute name="Title" value="ft/s"/></AxisUnit>
    </AxisX>
    <AxisY>
```

```

    <AxisTitle>
      <Attribute name="Title" value="The Y-Axis" />
      <Attribute name="TextColor" value="red" />
      <Attribute name="FontStyle" value="BOLD" />
    </AxisTitle>
    <AxisUnit><Attribute name="Title" value="temp" /></AxisUnit>
  </AxisY>
  <Data y="{6,5,7,1}">
    <Attribute name="LineColor" value="blue" />
    <Attribute name="LineWidth" value="2.0" />
    <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER" />
    <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE" />
  </Data>
  <Data y="{1,3,6,8}">
    <Attribute name="MarkerColor" value="darkblue" />
    <Attribute name="DataType" value="DATA_TYPE_MARKER" />
    <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE" />
  </Data>
</AxisXY>
</Chart>

```

## XML Example: Multiple Axes

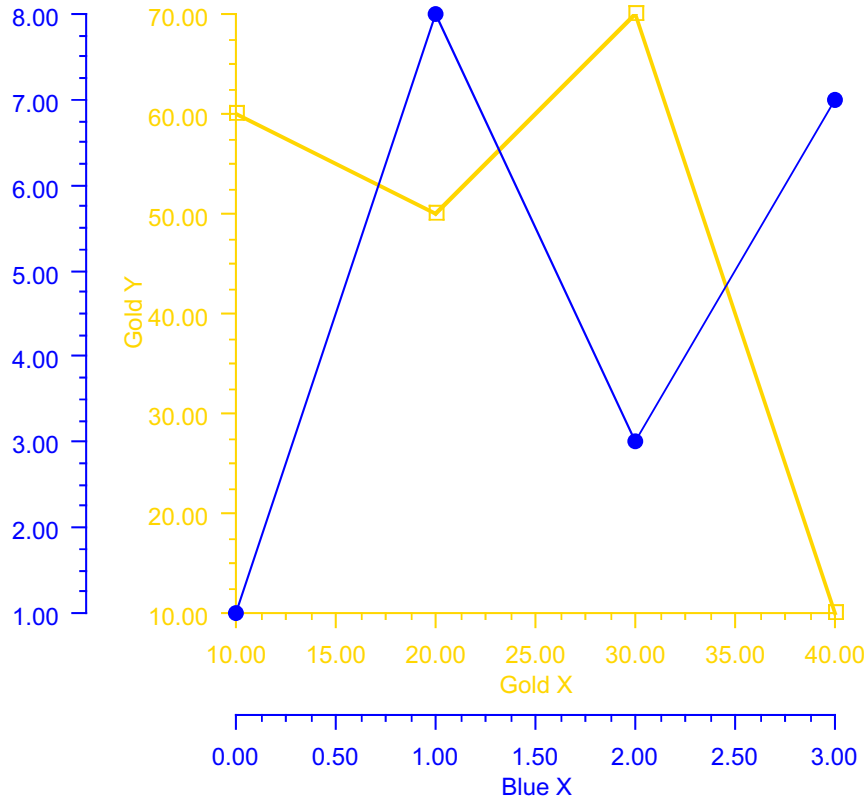


```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <Legend>
    <Attribute name="Paint" value="true"/>
    <Attribute name="Viewport" value="0.83,1.0,0.1,0.3"/>
  </Legend>
  <AxisXY>
    <Attribute name="Viewport" value="0.13,0.75,0.2,0.8"/>
    <Attribute name="LineColor" value="red"/>
    <Attribute name="MarkerColor" value="red"/>
    <Attribute name="TextColor" value="red"/>
    <Data x="10,20,30,40" y="60,50,70,10">
      <Attribute name="LineWidth" value="2.0"/>
      <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE"/>
      <Attribute name="Title" value="Red Axis"/>
      <ToolTip/>
    </Data>
  </AxisXY>
  <AxisXY>
    <Attribute name="Viewport" value="0.13,0.75,0.2,0.8"/>
    <Attribute name="LineColor" value="blue"/>
    <Attribute name="MarkerColor" value="blue"/>
    <Attribute name="TextColor" value="blue"/>
    <AxisX>
      <Attribute name="Type" value="AXIS_X_TOP"/>
    </AxisX>
    <AxisY>
      <Attribute name="Type" value="AXIS_Y_RIGHT"/>
    </AxisY>
    <Data y="1,8,3,7">
      <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE"/>
      <Attribute name="Title" value="Blue Axis"/>
      <ToolTip/>
    </Data>
  </AxisXY>
</Chart>

```

## XML Example: Axis Offset



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <AxisXY>
    <Attribute name="Viewport" value="0.3,0.9,0.1,0.7"/>
    <Attribute name="LineColor" value="gold"/>
    <Attribute name="MarkerColor" value="gold"/>
    <Attribute name="TextColor" value="gold"/>
    <AxisX><AxisTitle><Attribute name="Title" value="Gold X"/></AxisTi-
tle></AxisX>
    <AxisY><AxisTitle><Attribute name="Title" value="Gold Y"/></AxisTi-
```

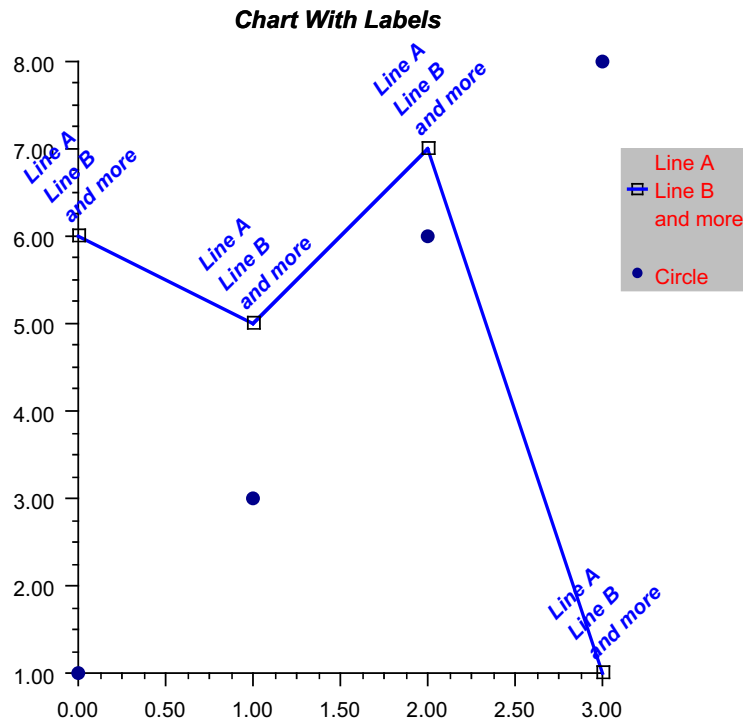


```

tle></AxisY>
    <Data x="10,20,30,40" y="60,50,70,10">
        <Attribute name="LineWidth" value="2.0"/>
        <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
        <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE"/>
        <Attribute name="Title" value="Gold Axis"/>
    </Data>
</AxisXY>
<AxisXY>
    <Attribute name="Cross" value="-0.75,-0.2"/>
    <Attribute name="Viewport" value="0.3,0.9,0.1,0.7"/>
    <AxisX><AxisTitle><Attribute name="Title" value="Blue X"/></AxisTi-
tle></AxisX>
    <AxisY><AxisTitle><Attribute name="Title" value="Blue Y"/></AxisTi-
tle></AxisY>
    <Attribute name="LineColor" value="blue"/>
    <Attribute name="MarkerColor" value="blue"/>
    <Attribute name="TextColor" value="blue"/>
    <Data y="1,8,3,7">
        <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER"/>
        <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE"/>
        <Attribute name="Title" value="Blue Axis"/>
    </Data>
</AxisXY>
</Chart>

```

## XML Example: Rotated Labels



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMML Chart//EN" "http://
www.vni.com/products/jmml/Chart.dtd">
<Chart>
  <ChartTitle>
    <Attribute name="Title">Chart With Labels</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <Legend>
    <Attribute name="Paint" value="true"/>
    <Attribute name="TextColor" value="red"/>
    <Attribute name="FillType" value="FILL_TYPE_SOLID"/>
  </Legend>
  <AxisXY>
    <Attribute name="ClipData" value="false"/>
    <Data y="{6,5,7,1}">
      <Attribute name="LabelType" value="LABEL_TYPE_TITLE"/>
    </Data>
  </AxisXY>
</Chart>
```

```

    <Attribute name="TextAngle" value="45" />
    <Attribute name="LineColor" value="blue" />
    <Attribute name="LineWidth" value="2.0" />
    <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER" />
    <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE" />
    <Attribute name="TextColor" value="blue" />
    <Attribute name="FontStyle" value="3" />
    <Attribute name="Title"><![CDATA[Line A

```

Line B

```
and more]]></Attribute>
```

```
</Data>
```

```
<Data y="{1,3,6,8}">
```

```
  <Attribute name="MarkerColor" value="darkblue" />
```

```
  <Attribute name="DataType" value="DATA_TYPE_MARKER" />
```

```
  <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE" />
```

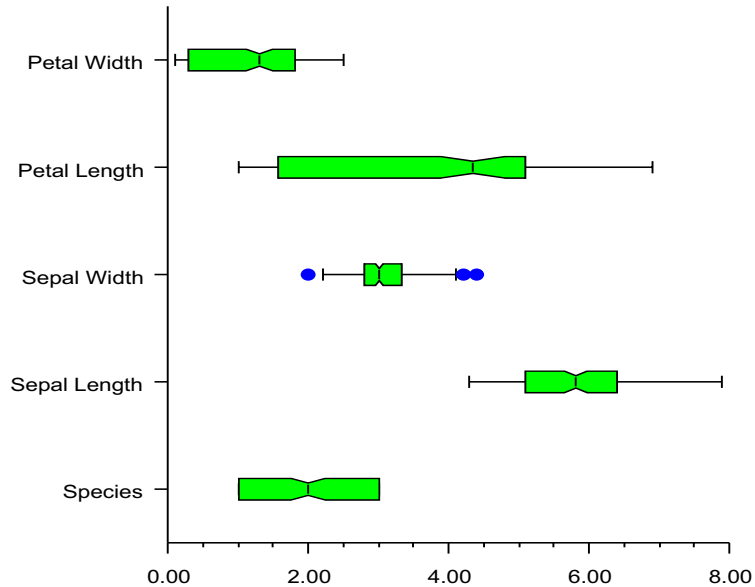
```
  <Attribute name="Title" value="Circle" />
```

```
</Data>
```

```
</AxisXY>
```

```
</Chart>
```

## XML Example: Box Plot



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <Array id="ozone">
    <Array>
      <NumberList>
        66.0, 52.0, 49.0, 64.0, 68.0, 26.0, 86.0, 52.0,
        43.0, 75.0, 87.0, 188.0, 118.0, 103.0, 82.0,
        71.0, 103.0, 240.0, 31.0, 40.0, 47.0, 51.0, 31.0,
        47.0, 14.0, 71.0
      </NumberList>
    </Array>
    <Array>
      <NumberList>
        61.0, 47.0, 196.0, 131.0, 173.0, 37.0, 47.0,
        215.0, 230.0, 69.0, 98.0, 125.0, 94.0, 72.0,
        72.0, 125.0, 143.0, 192.0, 122.0, 32.0, 114.0,

```

```

        32.0, 23.0, 71.0, 38.0, 136.0, 169.0
    </NumberList>
</Array>
<Array>
    <NumberList>
        152.0, 201.0, 134.0, 206.0, 92.0, 101.0, 119.0,
        124.0, 133.0, 83.0, 60.0, 124.0, 142.0, 124.0, 64.0,
        75.0, 103.0, 46.0, 68.0, 87.0, 27.0,
        73.0, 59.0, 119.0, 64.0, 111.0
    </NumberList>
</Array>
<Array>
    <NumberList>
        80.0, 68.0, 24.0, 24.0, 82.0, 100.0, 55.0, 91.0,
        87.0, 64.0, 170.0, 86.0, 202.0, 71.0, 85.0, 122.0,
        155.0, 80.0, 71.0, 28.0, 212.0, 80.0, 24.0,
        80.0, 169.0, 174.0, 141.0, 202.0
    </NumberList>
</Array>
<Array>
    <NumberList>
        113.0, 38.0, 38.0, 28.0, 52.0, 14.0, 38.0, 94.0,
        89.0, 99.0, 150.0, 146.0, 113.0, 38.0, 66.0, 38.0,
        80.0, 80.0, 99.0, 71.0, 42.0, 52.0, 33.0, 38.0,
        24.0, 61.0, 108.0, 38.0, 28.0
    </NumberList>
</Array>
</Array>
<Array id="months"><String value="May"/><String value="June"/><String
value="July"/><String value="August"/><String value="September"/></Array>
<Array id="grad"><Color value="yellow"/><Color value="lightblue"/><Color
value="lightblue"/><Color value="yellow"/></Array>

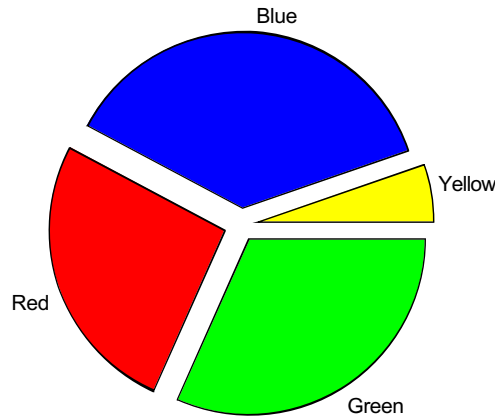
<ChartTitle><Attribute name="Title" value="Ozone Levels in Stanford by
Month"/></ChartTitle>
<AxisXY>
    <BoxPlot obs="#ozone">
        <Attribute name="BoxPlotType" value="BOXPLOT_TYPE_HORIZONTAL"/>
        <Attribute name="Labels" value="#months"/>
        <Attribute name="Notch" value="true"/>
        <Attribute name="ProportionalWidth" value="true"/>
        <Bodies>
            <Attribute name="FillType" value="FILL_TYPE_GRADIENT"/>
            <Attribute name="Gradient" value="#grad"/>
        </Bodies>
        <OutsideMarkers>
            <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_CIRCLE"/>
            <Attribute name="MarkerColor" value="purple"/>
        </OutsideMarkers>
    </BoxPlot>
</AxisXY>

```

```
<FarMarkers>
  <Attribute name="MarkerType" value="MARKER_TYPE_ASTERISK"/>
  <Attribute name="MarkerColor" value="red"/>
</FarMarkers>
<Whiskers>
  <Attribute name="MarkerColor" value="darkgreen"/>
</Whiskers>
</BoxPlot>
<AxisY>
  <AxisTitle><Attribute name="Title" value="Month"/></AxisTitle>
</AxisY>
<AxisX>
  <AxisTitle><Attribute name="Title" value="Ozone Level"/></AxisTitle>
  <Attribute name="TextFormat" value="#" />
</AxisX>
</AxisXY>
</Chart>
```

## XML Example: Pie Chart

*Pie Chart*

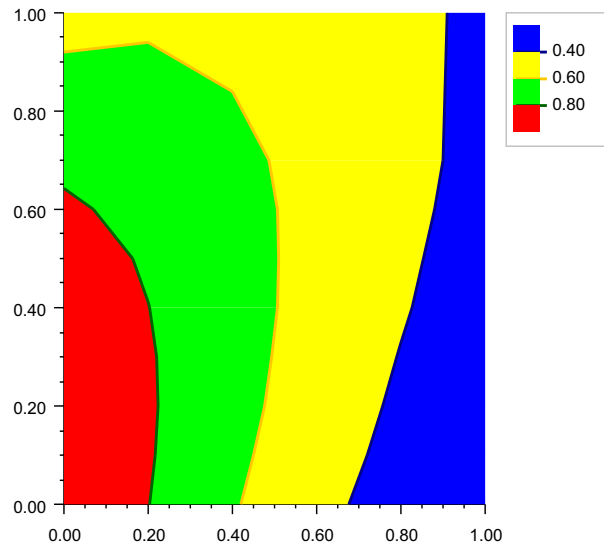


```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <ChartTitle>
    <Attribute name="Title">Pie Chart</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <Pie y="{6,5,7,1}">
    <Attribute name="LabelType" value="LABEL_TYPE_TITLE"/>
    <Attribute name="Explode" value="0.1"/>
    <PieSlice index="0">
      <Attribute name="FillColor" value="green"/>
      <Attribute name="Title" value="Green"/>
      <ToolTip/>
    </PieSlice>
    <PieSlice index="1">
      <Attribute name="FillColor" value="red"/>
      <Attribute name="Title" value="Red"/>
      <ToolTip/>
    </PieSlice>
    <PieSlice index="2">
```

```
        <Attribute name="FillColor" value="blue"/>
        <Attribute name="Title" value="Blue"/>
        <ToolTip/>
    </PieSlice>
    <PieSlice index="3">
        <Attribute name="FillColor" value="yellow"/>
        <Attribute name="Title" value="Yellow"/>
        <ToolTip/>
    </PieSlice>
</Pie>
</Chart>
```



## XML Example: Contour Chart



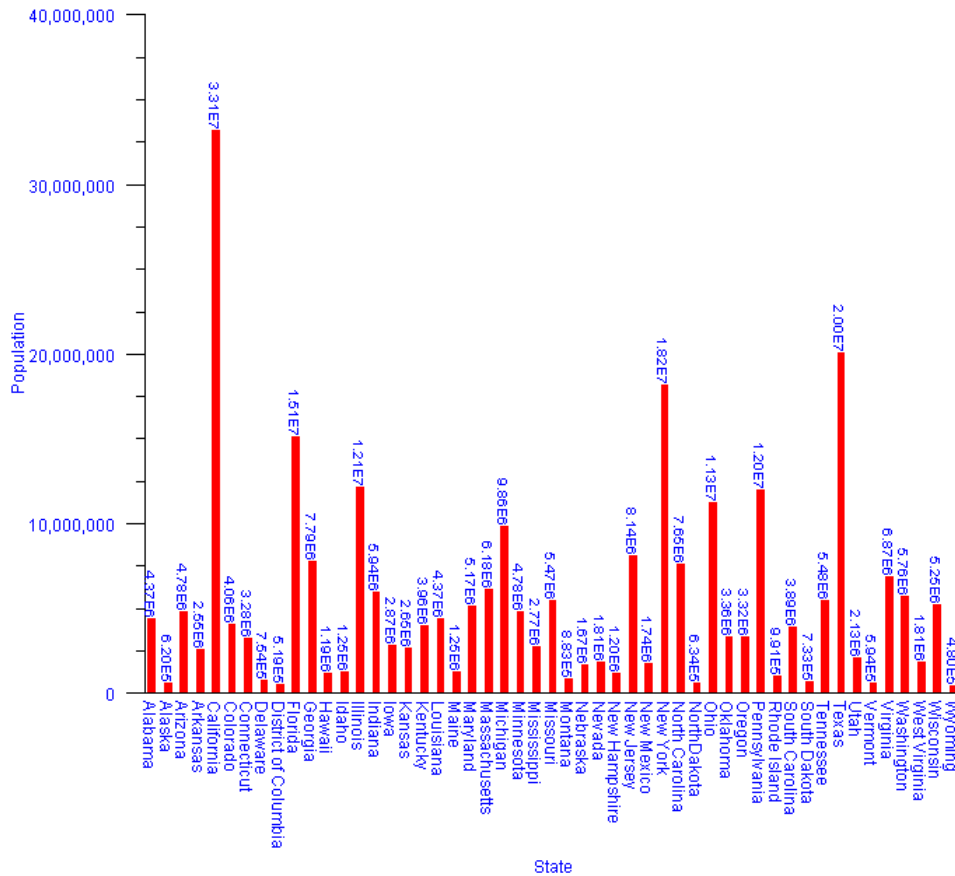
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMJS Chart//EN" "http://
www.vni.com/products/jmjsl/Chart.dtd">
<Chart>
  <Array id="x"><NumberList>5,3,1,2,4</NumberList></Array>
  <Array id="y"><NumberList>1,2,3,4,5</NumberList></Array>
  <Array id="z"><NumberList>26,28,27,29,25</NumberList></Array>

  <ChartTitle><Attribute name="Title" value="Scattered Data Contour Plot"/>
</ChartTitle>
  <AxisXY>
    <Contour xGrid="#x" yGrid="#y" zData="#z">
      <ContourLegend>
        <Attribute name="Paint" value="true"/>
      </ContourLegend>
      <ContourLevel index="5">
        <Attribute name="FillColor" value="green"/>
      </ContourLevel>
    </Contour>
    <Data x="#x" y="#y">
      <Attribute name="DataType" value="DATA_TYPE_MARKER"/>
      <Attribute name="MarkerType" value="MARKER_TYPE_FILLED_CIRCLE"/>
    </Data>
  </AxisXY>
</Chart>
```

```
        <Attribute name="MarkerColor" value="red"/>
    </Data>
</AxisXY>
</Chart>
```

## XML Example: Bar Chart of States

*Population of the United States*



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <!-- Data from http://www.census.gov/population/estimates/state/st-99-
3.txt -->
  <Array id="population">
    <NumberList>
      4369862,619500,4778332,2551373,33145121,4056133,3282031,753538,
      519000,15111244,7788240,1185497,1251700,12128370,5942901,2869413,
      2654052,3960825,4372035,1253040,5171634,6175169,9863775,4775508,
      2768619,5468338,882779,1666028,1809253,1201134,8143412,1739844,
```

```

        18196601,7650789,633666,11256654,3358044,3316154,11994016,
        990819,3885736,733133,5483535,20044141,2129836,593740,6872912,
        5756361,1806928,5250446,479602
    </NumberList>
</Array>
<Array id="states">
    <String value="Alabama"/>
    <String value="Alaska"/>
    <String value="Arizona"/>
    <String value="Arkansas"/>
    <String value="California"/>
    <String value="Colorado"/>
    <String value="Connecticut"/>
    <String value="Delaware"/>
    <String value="District of Columbia"/>
    <String value="Florida"/>
    <String value="Georgia"/>
    <String value="Hawaii"/>
    <String value="Idaho"/>
    <String value="Illinois"/>
    <String value="Indiana"/>
    <String value="Iowa"/>
    <String value="Kansas"/>
    <String value="Kentucky"/>
    <String value="Louisiana"/>
    <String value="Maine"/>
    <String value="Maryland"/>
    <String value="Massachusetts"/>
    <String value="Michigan"/>
    <String value="Minnesota"/>
    <String value="Mississippi"/>
    <String value="Missouri"/>
    <String value="Montana"/>
    <String value="Nebraska"/>
    <String value="Nevada"/>
    <String value="New Hampshire"/>
    <String value="New Jersey"/>
    <String value="New Mexico"/>
    <String value="New York"/>
    <String value="North Carolina"/>
    <String value="NorthDakota"/>
    <String value="Ohio"/>
    <String value="Oklahoma"/>
    <String value="Oregon"/>
    <String value="Pennsylvania"/>
    <String value="Rhode Island"/>
    <String value="South Carolina"/>
    <String value="South Dakota"/>
    <String value="Tennessee"/>

```

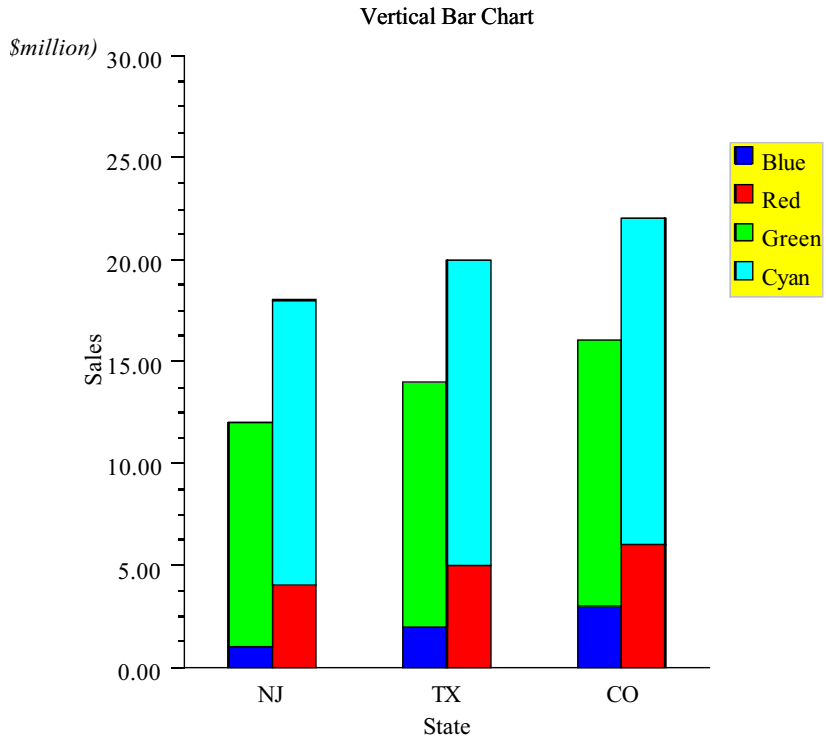
```

    <String value="Texas" />
    <String value="Utah" />
    <String value="Vermont" />
    <String value="Virginia" />
    <String value="Washington" />
    <String value="West Virginia" />
    <String value="Wisconsin" />
    <String value="Wyoming" />
</Array>

<Attribute name="TextColor" value="blue" />
<Background>
    <Attribute name="FillColor" value="white" />
    <Attribute name="FontSize" value="24" />
</Background>
<ChartTitle>
    <Attribute name="Title">Population of the United States</Attribute>
    <Attribute name="FontSize" value="24" />
    <Attribute name="FontStyle" value="3" />
</ChartTitle>
<AxisXY>
    <AxisY>
        <AxisTitle><Attribute name="Title" value="Population" /></AxisTitle>
        <Attribute name="TextFormat">###,###</Attribute>
    </AxisY>
    <AxisX>
        <AxisTitle><Attribute name="Title" value="State" /></AxisTitle>
        <AxisLabel><Attribute name="TextAngle" value="270" /></AxisLabel>
    </AxisX>
    <Bar y="#population">
        <Attribute name="BarType" value="BAR_TYPE_VERTICAL" />
        <Attribute name="Labels" value="#states" />
        <Attribute name="FillColor" value="red" />
        <Attribute name="FillOutlineType" value="FILL_TYPE_NONE" />
        <Attribute name="LabelType" value="LABEL_TYPE_Y" />
        <Attribute name="TextAngle" value="270" />
        <Attribute name="TextFormat" value="0.00E0" />
        <Attribute name="FontSize" value="10" />
    </Bar>
</AxisXY>
</Chart>

```

## XML Example: Bar Chart



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <Array id="foo">
    <Array>
      <Array><NumberList value="1,2,3"/></Array>
      <Array><NumberList value="4,5,6"/></Array>
    </Array>
    <Array>
      <Array><NumberList value="11,12,13"/></Array>
      <Array><NumberList value="14,15,16"/></Array>
    </Array>
  </Array>
  <Array id="states"><String value="NJ"/><String value="TX"/><String
value="CO"/></Array>
  <Array id="bg"><Color value="lightblue"/><Color value="lightblue"/><Color
value="yellow"/><Color value="yellow"/></Array>

  <Attribute name="FontName" value="Serif"/>

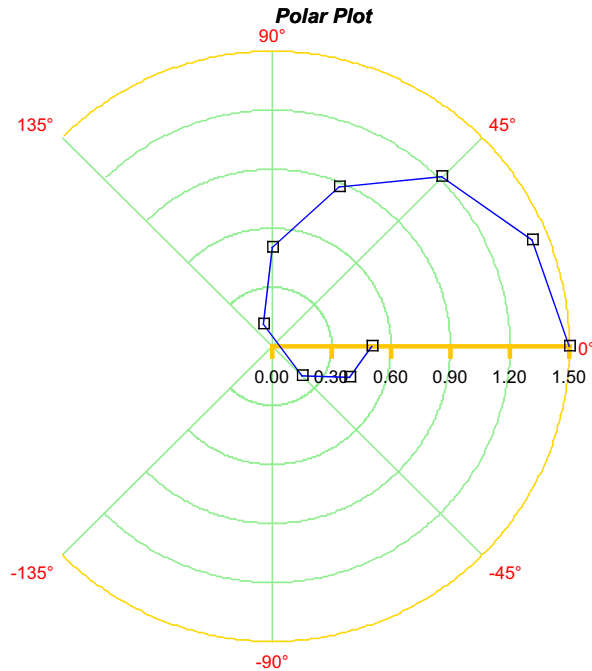
```

```

    <Attribute name="FontSize" value="14"/>
    <ChartTitle><Attribute name="Title" value="Vertical Bar Chart"/></
ChartTitle>
    <Background>
        <Attribute name="Gradient" value="#bg"/>
    </Background>
    <Legend>
        <Attribute name="Paint" value="true"/>
        <Attribute name="FillColor" value="yellow"/>
        <Attribute name="FillType" value="FILL_TYPE_SOLID"/>
    </Legend>
    <AxisXY>
        <AxisY>
            <AxisTitle><Attribute name="Title" value="Sales"/></AxisTitle>
            <AxisUnit>
                <Attribute name="Title" value="(in $million)"/>
                <Attribute name="FontStyle" value="ITALIC"/>
            </AxisUnit>
        </AxisY>
        <AxisX>
            <AxisTitle><Attribute name="Title" value="State"/></AxisTitle>
        </AxisX>
        <Bar x="{0,1,2}" y="#foo">
            <Attribute name="BarType" value="BAR_TYPE_VERTICAL"/>
            <Attribute name="Labels" value="#states"/>
            <BarSet index="{0,0}">
                <Attribute name="FillColor" value="blue"/>
                <Attribute name="Title" value="Blue"/>
            </BarSet>
            <BarSet index="{0,1}">
                <Attribute name="FillColor" value="red"/>
                <Attribute name="Title" value="Red"/>
            </BarSet>
            <BarSet index="{1,0}">
                <Attribute name="FillColor" value="green"/>
                <Attribute name="Title" value="Green"/>
            </BarSet>
            <BarSet index="{1,1}">
                <Attribute name="FillColor" value="cyan"/>
                <Attribute name="Title" value="Cyan"/>
            </BarSet>
        </Bar>
    </AxisXY>
</Chart>

```

## XML Example: Polar Plot



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Chart PUBLIC "-//Visual Numerics//DTD JMSL Chart//EN" "http://
www.vni.com/products/jmsl/Chart.dtd">
<Chart>
  <ChartTitle>
    <Attribute name="Title">Polar Plot</Attribute>
    <Attribute name="FontSize" value="14"/>
    <Attribute name="FontStyle" value="3"/>
  </ChartTitle>
  <Polar>
    <Attribute name="LineColor" value="gold"/>
    <AxisR>
      <Attribute name="LineColor" value="orange"/>
      <AxisRLine>
        <Attribute name="LineWidth" value="3"/>
      </AxisRLine>
      <AxisRMajorTick>
        <Attribute name="LineWidth" value="3"/>
      </AxisRMajorTick>
    </AxisR>
  </Polar>
</Chart>
```



```

        </AxisRMajorTick>
    </AxisR>
    <AxisTheta>
        <Attribute name="Window" value="{ -2.35619449019, 2.35619449019 }" />
        <Attribute name="Number" value="7" />
        <Attribute name="TextColor" value="red" />
    </AxisTheta>
    <GridPolar>
        <Attribute name="LineColor" value="lightgreen" />
    </GridPolar>
    <Data x="{ 1.50, 1.42, 1.21, 0.88, 0.50, 0.12, -0.21, -0.42, -0.50 }"
        y="{ 0.00, 0.39, 0.79, 1.18, 1.57, 1.96, 2.36, 2.75, 3.14 }" >
        <Attribute name="LineColor" value="blue" />
        <Attribute name="DataType" value="DATA_TYPE_LINE|DATA_TYPE_MARKER" />
        <Attribute name="MarkerType" value="MARKER_TYPE_HOLLOW_SQUARE" />
        <Attribute name="Title" value="Blue Line" />
        <ToolTip/>
    </Data>
</Polar>
</Chart>

```

# Chapter 5: Actions

JMSL supports the following actions:

- [“Picking” on page 169](#)
- [“Zoom” on page 172](#)
- [“Printing” on page 178](#)
- [“Serialization” on page 180](#)

---

## Picking

The pick mechanism allows `MouseEvent`s to be associated with chart nodes. The pick mechanism follows the Java listener pattern.

The class that is to handle the pick events implements the `com.imsl.chart.PickListener` interface. This requires the implementation of the method `pickPerformed(PickEvent)`.

The `PickListener` is then added to a chart node, using the node's `addPickListener` method.

A pick is fired by calling the method `pick(MouseEvent)` in the top-level `Chart` node. Normally this is done in response to a mouse click. Mouse clicks can be detected by implementing the `MouseListener` interface and adding it to the chart's container.

## Example

In this example, when a bar is clicked it toggles its color between blue and red.

A `MouseListener` is implemented as an inner class. Its `mouseClicked` method calls `chart.pick(MouseEvent)`.

The class `SamplePick` implements the `PickListener` interface. It is not an inner class only because of the size of the `pickPerformed` method. The pick listener is added to the chart node `bar`. This means that it is active for that node and its children but is not active for the axes and other nodes.

```
import com.imsl.chart.*;
import java.awt.Color;

public class SamplePick extends JFrameChart implements PickListener {
    static private final double x[] = {0, 1, 2, 3};
    static private final double y[] = {4, 1, 2, 5};

    public SamplePick() {
        // Create a bar chart
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        Bar bar = new Bar(axis, x, y);
        bar.setBarType(Bar.BAR_TYPE_VERTICAL);
        bar.setLabels(new String[]{"A", "B", "C"});
        bar.setFill(Color.blue);

        // Add the pick listener to the bar
        bar.addPickListener(this);

        // Add the mouse listener
        addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent event) {
                getChart().pick(event);
            }
        });
    }

    public void pickPerformed(PickEvent event) {
        // Track the number of hits on this bar
        ChartNode node = event.getNode();
        int count = node.getIntegerAttribute("pick.count", 0);
        count++;
        node.setAttribute("pick.count", new Integer(count));

        // Change the bar's color depending on the number of times
        // it has been picked.
        Color color = ((count%2==0) ? Color.blue : Color.red);
    }
}
```

```
        node.setFillColor(color);
        node.setLineColor(color);
        repaint();
    }

    public static void main(String argv[]) {
        new SamplePick().setVisible(true);
    }
}
```

---

## Zoom

JMSL provides the functionality to implement a variety of zoom policies instead of implementing a single specific zoom policy. The following example shows how to build a zoom policy that draws a rubber-band box in a scatter plot and zooms both the  $x$ -axis and  $y$ -axis to correspond to the region selected. Zoom out is also implemented.

This code can be used as the basis of an implementation of other zoom policies. For example, with a function plot one might want to only change the  $x$ -axis in response to a zoom and leave the  $y$ -axis alone. In other situations it may be desirable to zoom in or out a fixed amount around the location of a mouse click.

### Example

In the constructor:

- A scatter plot is created,
- Zoom items are added to the main menu,
- A `MouseListener` is added to the panel. A `JFrameChart` contains a number of components, including a `JPanel` in which the chart is drawn. *It is important that the `MouseListener` be added to the component in which the chart is drawn*, so that the `MouseEvent` coordinates are the same as the device coordinates of the chart.

A mouse press event starts the zoom. In response to the event a `MouseMotionListener` is added to the panel to listen for mouse drag events. The location of this initial event is also recorded.

A mouse drag event resizes the area of the zoom. In response, the old rubber-band box is erased and a new box is drawn. The new box has the initial mouse press location as one corner and this drag event location as the opposite corner. The location of this event is stored in `(xLast,yLast)`.

A mouse release event completes the zoom. In response,

- The `MouseListener` is removed from the panel,
- The existing `Window` attributes for the  $x$ -axis and  $y$ -axis are saved on a stack. They may be used later to zoom out.
- The new `Window` is computed. The `Window` is the range, in user coordinates, along an axis. The method `Axis.mapDeviceToUser` is used to map the device (pixel) coordinates to the user coordinate space. The new windows are constrained to be inside of the existing window.
- Autoscaling is turned off, so that the new `Window` values will be used without further modification.
- The chart is redrawn with the new windows.

The menu items, added by the constructor to the main menu, can be used to zoom out. The added menu items added "this" as an `ActionListener`, so the `actionPerformed` method is called when these menu items are selected. The `actionPerformed` method implements "Out" by popping previous `Window` attribute values off a stack. It implements "Restore" by re-enabling autoscaling.

```
import com.imsl.chart.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Stack;
import javax.swing.JMenu;
import javax.swing.JMenuItem;

public class SampleZoom extends JFrameChart
    implements MouseListener, MouseMotionListener, ActionListener {
    private javax.swing.JPanel panel;
    private AxisXY axis;
    private int xFirst, yFirst;
    private int xLast, yLast;
    private Stack stack;

    public SampleZoom() {
        // create a scatter plot with some random points
        Chart chart = getChart();
        axis = new AxisXY(chart);
        int n = 1000;
        double x[] = new double[n];
        double y[] = new double[n];
        for (int k = 0; k < n; k++) {
            x[k] = 10.*Math.random();
            y[k] = 10.*Math.random();
        }
    }
}
```

```

Data data = new Data(axis, x, y);
data.setDataType(Data.DATA_TYPE_MARKER);

// add menu "Zoom" menu
JMenu jMenuItemZoom = new JMenu("Zoom");
jMenuItemZoom.setMnemonic('Z');
addMenuItem(jMenuItemZoom, "Out");
addMenuItem(jMenuItemZoom, "Restore");
getJMenuBar().add(jMenuItemZoom);

// save x-axis and y-axis Window attributes on this stack.
// they are used to zoom out one level.
stack = new Stack();

// listen for mouse press events
panel = getPanel();
panel.addMouseListener(this);
}

/**
 * Add a menu item
 */
private void addMenuItem(JMenu jMenuItemZoom, String title) {
    JMenuItem jMenuItem = new JMenuItem(title);
    jMenuItem.setActionCommand(title);
    jMenuItem.setMnemonic(title.charAt(0));
    jMenuItemZoom.add(jMenuItem);
    jMenuItem.addActionListener(this);
}

/**
 * A mouse press starts the zoom
 * Record location of this point and listen for drag (motion) events.
 */
public void mousePressed(MouseEvent event) {
    xFirst = xLast = event.getX();
    yFirst = yLast = event.getY();
    panel.addMouseMotionListener(this);
}

/**
 * A mouse release ends the zoom.
 * Stop listening for drag events and update the
 * Window attribute in the x and y axes.
 */
public void mouseReleased(MouseEvent event) {
    panel.removeMouseMotionListener(this);

    // ignore degenerate zooms

```

```

    if (xFirst==xLast || yFirst==yLast) {
        repaint();
        return;
    }

    // get window and convert to user coordinates
    double windowX[] = (double[])axis.getAxisX().getWindow();
    double windowY[] = (double[])axis.getAxisY().getWindow();

    // save the windows on the stack (for zoom out option)
    stack.add(windowX);
    stack.add(windowY);

    // get user coordinate of left-upper corner of the box
    // limit it to stay inside current window
    double boxLU[] = new double[2];
    int x = Math.min(xFirst, xLast);
    int y = Math.min(yFirst, yLast);
    axis.mapDeviceToUser(x, y, boxLU);

    // get user coordinate of right-lower corner of the box
    // limit it to stay inside current window
    double boxRL[] = new double[2];
    x = Math.max(xFirst, xLast);
    y = Math.max(yFirst, yLast);
    axis.mapDeviceToUser(x, y, boxRL);

    // set axis window to range of rubber-band box
    // and disable autoscale to force use of window settings
    axis.setAutoscaleInput(ChartNode.AUTOSCALE_OFF);
    double xa = Math.max(windowX[0], boxLU[0]);
    double xb = Math.min(windowX[1], boxRL[0]);
    double ya = Math.max(windowY[0], boxRL[1]);
    double yb = Math.min(windowY[1], boxLU[1]);
    axis.getAxisX().setWindow(xa, xb);
    axis.getAxisY().setWindow(ya, yb);

    // force redraw of the chart
    repaint();
}

/**
 * A drag event continues the zoom.
 * Erase the old rubber-band box and draw a new one.
 * Also keep track of the location of this event.
 */
public void mouseDragged(MouseEvent event) {
    // erase previous rectangle
    Graphics g = panel.getGraphics();

```



```

    g.setXORMode(Color.cyan); // complement of drawing color red
    drawBox(g);

    // draw new rectangle
    xLast = event.getX();
    yLast = event.getY();
    g.setPaintMode();
    g.setColor(Color.red);
    drawBox(g);
}

public void mouseMoved(MouseEvent event) {}
public void mouseEntered(MouseEvent event) {}
public void mouseClicked(MouseEvent event) {}
public void mouseExited(MouseEvent event) {}

/**
 * Draw a box with (xFirst,yFirst) and (xLast,yLast) as its corners.
 */
private void drawBox(Graphics g) {
    int x = Math.min(xFirst, xLast);
    int y = Math.min(yFirst, yLast);
    int w = Math.abs(xLast - xFirst);
    int h = Math.abs(yLast - yFirst);
    g.drawRect(x, y, w, h);
}

/**
 * Handle menu item events.
 * Command "Out" zooms out one level.
 * Command "Restore" undoes all zooms.
 */
public void actionPerformed(ActionEvent actionEvent) {
    String command = actionEvent.getActionCommand();
    if (command.equals("Out")) {
        try {
            // zoom out by restoring window settings from the stack
            axis.getAxisY().setWindow((double[])stack.pop());
            axis.getAxisX().setWindow((double[])stack.pop());
        } catch (java.util.EmptyStackException e) {
            // no more levels to zoom out
            // restore original setting by turning on autoscale
            axis.setAutoscaleInput(ChartNode.AUTOSCALE_DATA);
        }
    } else if (command.equals("Restore")) {
        // restore original setting by turning on autoscale an empty stack
        axis.setAutoscaleInput(ChartNode.AUTOSCALE_DATA);
        stack.empty();
    }
}

```

```
        // force redraw of the chart
        repaint();
    }

    public static void main(String argv[]) {
        new SampleZoom().setVisible(true);
    }
}
```

---

## Printing

### Printing from JFrameChart

The `JFrameChart` class, used to build most of the examples in this manual, includes a print option under the file menu. This option prints the chart as large as possible, without distortion, and centered on the page.

### Printable Interface

The Java `Printable` interface is used to print a single page from Java. It is implemented by the class `Chart`. The following code fragment shows how to print a chart using its `Printable` interface and `PrinterJob`.

```
public void print() {
    PrinterJob printJob = PrinterJob.getPrinterJob();
    printJob.setPrintable(chart);
    if (printJob.printDialog()) {
        try {
            printJob.print();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

The `PrinterJob.printDialog()` method displays a dialog box that may contain a “Property Sheet.” This property sheet is supplied by the hardware printer driver, not by Java. User changes to property sheet values are not reflected back to Java. Specifically, while a property sheet may allow the user to select landscape or portrait mode, these settings are not reflected back to the `PrinterJob` object. The correct way to set the orientation is by using the `PrinterJob.pageDialog(PageFormat)` method. For more details see bug report 4311283 on Sun’s Java site.

### Pageable

The Java `Pageable` interface is used to print a complex document from Java. The method `Chart.paintChart(Graphics)` can be used to implement a `Pageable` interface. The following method is the implementation of the `Printable` interface in `Chart`. It uses the method `Chart.paintChart(Graphics)`.

Note that the chart is drawn to fit the Component associated with the chart. The `getScreenSize` method returns the size of this component. The following method scales the drawing to compensate for the difference in size between the screen and the paper.

```
public int print(Graphics graphics, PageFormat pageFormat, int param)
throws PrinterException {
    if (param >= 1) return Printable.NO_SUCH_PAGE;

    // translate, so we do not clip on the left
    graphics.translate((int)pageFormat.getImageableX(),
        (int)pageFormat.getImageableY());

    // scale to fill the page
    double dw = pageFormat.getImageableWidth();
    double dh = pageFormat.getImageableHeight();
    Dimension screenSize = getScreenSize();
    double xScale = dw / screenSize.width;
    double yScale = dh / screenSize.height;
    double scale = Math.min(xScale,yScale);

    // center the chart on the page
    double tx = 0.0;
    double ty = 0.0;
    if (xScale > scale) {
        tx = 0.5*(xScale-scale)*screenSize.width;
    } else {
        ty = 0.5*(yScale-scale)*screenSize.height;
    }
    ((Graphics2D)graphics).translate(tx, ty);
    ((Graphics2D)graphics).scale(scale, scale);

    Dimension bounds = new Dimension((int)dw, (int)dh);
    paint(new Draw(graphics,bounds));
    return Printable.PAGE_EXISTS;
}
```

---

## Serialization

The chart tree can be serialized into a stream of bytes and recreated from that stream.

Serialization is used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI). See Sun's Java Web site (<http://java.sun.com/j2se/1.3/docs/guide/serialization/>) for more details about serialization in general.

*Serialization of JMSL classes may not be compatible with future JMSL releases.* The current serialization support is appropriate for short term storage or RMI between applications running the same version of JMSL.

To serialize a chart tree, create an `ObjectOutputStream` and then use its `writeObject` method to write out the `Chart` object. The chart tree can be reconstructed from the stream by creating a `ObjectInputStream` and using its `readObject` method to read the `Chart` object.

The chart tree cannot be serialized if it contains any non-serializable objects, such as attribute values. All of the JMSL chart nodes are serializable. Image objects are not serializable, but `ImageIcon` objects are serializable. Standard Java Paint objects, such as `TexturePaint` objects, are not serializable. But the Paint objects generated by the JMSL class `FillPaint` are serializable.

### Example

When the following code is executed without any arguments, it creates a chart and serializes it to the file `sample.dat`. When it is executed with the single argument `-read`, it reads the file `sample.dat` and recreates the tree.

```
import com.imsl.chart.*;
import java.io.*;
import java.awt.*;
import javax.swing.ImageIcon;

public class SampleSerialization extends JFrameChart {

    private void createChart() {
        Chart chart = getChart();
        AxisXY axis = new AxisXY(chart);
        chart.getBackground().setFillType(ChartNode.FILL_TYPE_PAINT);
        Paint paint = FillPaint.checkerboard(24, Color.yellow, Color.orange);
        chart.getBackground().setFillPaint(paint);
        double y[] = {4, 6, 2, 1, 8};
```

```

        Data data = new Data(axis, y);
        data.setDataType(Data.DATA_TYPE_LINE);
        data.setLineColor("blue");
    }

    static private java.awt.Image loadImage(String name) {
        return new ImageIcon(Data.class.getResource(name)).getImage();
    }

    private void writeChart() throws IOException {
        FileOutputStream fos = new FileOutputStream("sample.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(getChart());
        oos.flush();
        fos.close();
    }

    private Chart readChart() throws IOException, ClassNotFoundException {
        FileInputStream fis = new FileInputStream("sample.dat");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Chart chart = (Chart)ois.readObject();
        fis.close();
        return chart;
    }

    public static void main(String argv[]) throws Exception {
        if (argv.length == 0) {
            SampleSerialization sample = new SampleSerialization();
            sample.createChart();
            sample.writeChart();
            System.exit(0);
        } else if (argv[0].equals("-read")) {
            SampleSerialization sample = new SampleSerialization();
            Chart chart = sample.readChart();
            sample.setChart(chart);
            sample.setVisible(true);
        } else {
            System.out.println("usage: java SampleSerialization [-read]");
        }
    }
}

```



# Appendix A: Web Server Application

A JMSL chart can be returned as an image from a Web server using a servlet. The mechanism depends on the version of the Java runtime library being used.

With JDK 1.4 or later, images are rendered using the standard Java class `javax.imageio.ImageIO`. This class can be used on a headless server—that is, a server running without graphical interfaces. While most desktop Unix systems run the X Windows System, Unix servers are often run without it. Java runs in a headless mode if the system property `java.awt.headless` is set to `true`.

With JDK 1.3 or earlier, the `ImageIO` class cannot be used because it does not exist in older versions of Java. Instead the Java Advanced Imaging (JAI) Toolkit is used. *JAI is not included with JMSL*. It is a standard Java extension that can be downloaded, for free, from Sun's JAI download site. Also, it requires that *a windowing system must be running*. With Microsoft Windows this is not usually a problem. A headless server cannot be used; on Unix, X Windows, or its equivalent, must be running.

---

## Overview—Web Servlets

Java Server Pages (JSP) and servlets are the standard mechanism for adding Java functionality to Web servers.

Adding charts to a Web page is complicated by the fact that in response to a request a Web server can return *either* a text HTML page *or* a bitmap image. The response to the initial request is an HTML page that contains an image tag. When the browser parses the HTML page and finds the image tag, it makes a second request to the Web server. The Web server then returns the bitmap image in response to this second request.

The JMSL classes `JspBean` and `ChartServlet` are used to coordinate this process. During the first request, one or more JMSL chart trees are constructed and stored in the `HttpSession` object, obtained from the `HttpServletRequest` object, using the



method `JspBean.registerChart`. The method `Jsp.getImageTag` is used to construct an HTML image tag with an identifier for the chart in the session object. This image tag refers to the servlet `ChartServlet`. The second browser request is therefore directed to the `ChartServlet`, which uses the identifier to retrieve the chart from the session object. The Java Advanced Imaging library is used by the servlet to render the chart as a PNG image, which is returned to the browser.

The use of the session object to hold the chart requires that the user's browser allow cookies, used to track sessions across requests.

---

## Generating a Chart with a JSP

This example consists of a JSP working with a bean (`pieChart`) that contains a `JspBean`. The HTML image tag is the value of the `imageTag` property from `pieChart`. The following is the JSP.

```
1 <%@page contentType="text/html"%>
2 <jsp:useBean id="pieChart" scope="page" class="SampleChartBean"/>
3 <html>
4 <head><title>Pie Chart</title></head>
5 <body>
6 <h1>Pie Chart</h1>
7 <jsp:setProperty name="pieChart" property="*" />
8 <% pieChart.createChart(request); %>
9 <jsp:getProperty name="pieChart" property="imageTag" />
10 </body>
11 </html>
```

In the above JSP, line 1 flags that an HTML page is being generated. Line 2 creates the object `pieChart`, which is an instance of `SampleChartBean`. This object has page scope, so it is discarded after the page has been generated. Line 7 sets attributes in `pieChart` using parameters in the request. Attributes are set in `pieChart` using its setter methods, following the usual pattern for Java beans. Line 8 calls the `createChart` method to create the chart tree and store it in the session. This is Java code enclosed in the special delimiters. Line 9 inserts the HTML image tag obtained by calling `pieChart.getImageTag()`.

The class `SampleChartBean` uses a `JspBean`. It creates a pie chart and registers it with the session obtained from the request. The `getImageTag()` method returns an HTML tag which refers to the `ChartServlet` servlet that returns the chart image to the browser.

```

import com.imsl.chart.*;
import java.awt.Color;

public class SampleChartBean {
    private JspBean bean;

    public SampleChartBean() {
        bean = new JspBean();
        bean.setSize(300, 300);
    }

    public void createChart(javax.servlet.http.HttpServletRequest request) {
        Chart chart = new Chart();
        bean.registerChart(chart, request);

        double y[] = {35., 20., 30., 40.};
        Pie pie = new Pie(chart, y);
        pie.setLabelType(pie.LABEL_TYPE_TITLE);
        pie.setFillOutlineColor(Color.blue);

        PieSlice[] slice = pie.getPieSlice();

        slice[0].setFillColor(Color.red);
        slice[0].setTitle("Red");

        slice[1].setFillColor(Color.blue);
        slice[1].setTitle("Blue");
        slice[1].setFillOutlineColor(Color.yellow);

        slice[2].setFillColor(Color.black);
        slice[2].setTitle("Black");
        slice[2].setExplode(0.3);

        slice[3].setFillColor(Color.yellow);
        slice[3].setTitle("Yellow");
    }

    public String getImageTag() {
        return bean.getImageTag();
    }
}

```

---

## Generating Charts with Client-side Imagemaps

Client-side image maps allow regions of an image to be linked to a URL. Clicking on such a region behaves the same as clicking on a hypertext link. This can be used to create a degree of interactivity. Target regions are created for chart nodes that have the HREF attribute defined.

In this example the sample pie chart as above is generated, but now when the user clicks on a slice the selected slice is exploded out. Now `pieChart` is an instance of `SampleImageBean`. The JSP gets the `pieChart`'s `imageMap` property to create a map tag in the HTML code.

```
<%@page contentType="text/html"%>
<jsp:useBean id="pieChart" scope="page" class="SampleImagemapBean" />
<html>
<head><title>Pie Chart Imagemap</title></head>
<body>
<h1>Pie Chart Imagemap</h1>
<jsp:setProperty name="pieChart" property="*" />
<% pieChart.createChart(request); %>
<jsp:getProperty name="pieChart" property="imageMap" />
<jsp:getProperty name="pieChart" property="imageTag" />
</body>
</html>
```

The code for `SampleImagemapBean` is similar to the code for the previous `SampleChartBean`, but with the HREF attribute defined in each slice. The reference is back to `SampleImagemapJSP`, but with the `explode` parameter defined. The second change is the addition of a setter method for `explode`. The line

```
<jsp:setProperty name="pieChart" property="*" />
```

in the JSP file sets properties in the `pieChart` from parameters in the request. If `explode` is defined in the HTTP request, then the JSP calls this setter method.

```
import com.imsl.chart.*;
import java.awt.Color;
```

```
public class SampleImagemapBean extends JspBean {
    private JspBean bean;
    private int     explode;

    public SampleImagemapBean() {
        bean = new JspBean();
        bean.setSize(300,300);
        bean.setCreateImageMap(true);
    }
}
```

```

public void createChart(javax.servlet.http.HttpServletRequest request) {
    Chart chart = new Chart();

    double y[] = {35., 20., 30., 40.};
    Pie pie = new Pie(chart, y);
    pie.setLabelType(pie.LABEL_TYPE_TITLE);
    pie.setFillOutlineColor(Color.blue);

    PieSlice slice[] = pie.getPieSlice();

    slice[0].setFillColor(Color.red);
    slice[0].setTitle("Red");

    slice[1].setFillColor(Color.blue);
    slice[1].setTitle("Blue");
    slice[1].setFillOutlineColor(Color.yellow);

    slice[2].setFillColor(Color.black);
    slice[2].setTitle("Black");

    slice[3].setFillColor(Color.yellow);
    slice[3].setTitle("Yellow");

    for (int k = 0; k < slice.length; k++) {
slice[k].setHREF("SampleImagemapJSP.jsp?explode="+k);
    }

    try {
        slice[explode].setExplode(0.3);
    } catch (Exception e) {
        // ignore out of range values for explode
    }

    bean.registerChart(chart, request);
}

public void setExplode(int explode) {
    this.explode = explode;
}

```

```
public String getImageTag() {  
    return bean.getImageTag();  
}  
  
public String getImageMap() {  
    return bean.getImageMap();  
}  
}
```

---

## Servlet Deployment

A Web server that supports Java servlets will generally have a directory of Web applications, one per directory. In each Web application subdirectory there can be a WEB-INF directory containing the Java classes. This directory contains the subdirectory classes, containing individual class files, and the subdirectory lib, containing JAR files. For the above example, the file structure would be as follows:

```
webapps/  
  application_name/  
    SampleChartJSP.jsp  
    SampleImagemapJSP.jsp  
    WEB-INF/  
      web.xml  
      classes/  
        SampleChartBean.class  
        SampleImagemapBean.class  
      lib/  
        jmsl.jar  
        jai_core.jar  
        jai_codec.jar
```

The file `jmsl.jar` contains the JMSL classes. The files `jai_core.jar` and `jai_codec.jar` contain the JAI classes.



# Appendix B: Writing a Chart as a Bitmap Image File

A JMSL chart can be saved as an image file in two ways: either using the Java `ImageIO` class or the Java Advanced Imaging (JAI) API.

## Using the `ImageIO` Class

The `ImageIO` class was added to Java in JDK1.4. The JAI, described in the next section, can be used with older versions of Java.

The chart tree is constructed in the usual manner. Here a method called `createChart` is used to create a simple chart, with the null-argument `Chart` constructor.

The bitmap is generated by creating a `BufferedImage` object and painting the chart into the buffered image using the graphics object from the buffered image. The buffered image is written to a file using the `ImageIO.write` method.

This method of creating bitmaps does not require that a windowing system be running. The argument `-Djava.awt.headless=true` can be used with the `java` command from JDK1.4, or later, to run Java in a "headless" mode.

```
import com.imsl.chart.*;
import java.awt.image.BufferedImage;
import java.io.File;

public class SampleImageIO {
    public static void main(String argv[]) throws java.io.IOException {
        Chart chart = createChart();
        chart.setScreenSize(new java.awt.Dimension(500,500));

        BufferedImage bi = new BufferedImage(500, 500,
        BufferedImage.TYPE_4BYTE_ABGR_PRE);
        chart.paintChart(bi.createGraphics());
    }
}
```



```

        File file = new File("SampleImageIO.png");
        javax.imageio.ImageIO.write(bi, "PNG", file);
    }

    static Chart createChart() {
        Chart chart = new Chart();
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * Math.PI/(npoints-1);
        double x[] = new double[npoints];
        double y[] = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++){
            x[i] = i * dx;
            y[i] = Math.sin(x[i]);
        }
        new Data(axis, x, y);
        return chart;
    }
}

```

## Using the Java Advanced Imaging API

A JMSL chart can be saved as an image file by generating a `java.awt.Image` object and then rendering the image using the Java Advanced Imaging (JAI) API. The example below shows the steps needed.

*JAI is not included with JMSL Numerical Library.* It is a standard Java extension that can be downloaded, for free, from Sun's JAI download site.

The first step is to create a `java.awt.Frame` object. A `Frame` object is required because the implementation of the drawing code is based on the code used to write to the screen. Because a `Frame` object is required, *a windowing system must be running* on the system. The size of the frame, here 500 by 500, determines the size of the bitmap image.

The chart tree is then constructed in the usual manner. Here a method called `createChart` is used to create a simple chart, with the `Frame` object passed to the `Chart` constructor.

An `Image` object is created using the `paintImage` method in `Chart`.

The create method in `javax.media.jai.JAI` can now be used to generate the file. The first call (with `AWTImage`) is used to generate a JAI object from the AWT Image object. The second call (with `filestore`) actually writes the file. In this case we write a PNG format file called `SampleJai.png`. JAI can be used to generate image files in other formats, but PNG is usually the best for charts.

The last step is to exit explicitly. The program does not automatically exit because of the existence of a user thread, implicitly created to handle the frame.

```
import com.imsl.chart.*;
import java.awt.*;
import javax.media.jai.*;

public class SampleJai {
    public static void main(String argv[]) {
        Frame frame = new Frame();
        frame.show();
        frame.setSize(500, 500);
        frame.setVisible(false);
        Chart chart = createChart(frame);

        Image image = frame.createImage(500, 500);
        chart.paintChart(image.getGraphics());
        RenderedOp im = JAI.create("AWTImage", image);
        JAI.create("filestore", im, "SampleJai.png", "PNG");
        System.exit(0);
    }

    static Chart createChart(Component component) {
        Chart chart = new Chart(component);
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * Math.PI/(npoints-1);
        double x[] = new double[npoints];
        double y[] = new double[npoints];

        // Generate some data
        for (int i = 0; i < npoints; i++){
            x[i] = i * dx;
            y[i] = Math.sin(x[i]);
        }
        new Data(axis, x, y);
        return chart;
    }
}
```

## Using the Scalable Vector Graphics (SVG) API

Scalable Vector Graphics (SVG) is an XML vocabulary for vector graphics.

JMSL's SVG support is based on the Apache Batik toolkit. *Batik is not included with JMSL Numerical Library.* It can be downloaded, for free, from the Apache Batik site. The `batik.jar` file needs to be added to the CLASSPATH.

Once `batik.jar` is installed, a JMSL chart can be saved as an SVG file by using the `Chart.writeSVG` method.

SVG can also be create from a `JFrameChart` window. Its File | SaveAs menu allows the chart to be saved as either a PNG image or as an SVG file. The PNG option is active only if J2SE 1.4 or later is being used, or if the JAI classes are in the CLASSPATH. The SVG option is active only if `batik.jar` is in the CLASSPATH.

The following example generates SVG output:

---

**NOTE:** The output encoding format is set to UTF-8. This is done because SVG renders do not support all encodings. The UTF-8 is the most general encoding.

---

```
import com.imsl.chart.*;
import java.io.*;
import java.awt.*;

public class SampleSVG {
    public static void main(String argv[]) throws java.io.IOException {
        Frame frame = new Frame();
        frame.show();
        frame.setSize(500, 500);
        frame.setVisible(false);
        Chart chart = createChart(frame);

        OutputStream os = new FileOutputStream("SampleSVG.svg");
        Writer writer = new OutputStreamWriter(os, "UTF-8");
        chart.writeSVG(writer, true);

        System.exit(0);
    }

    static Chart createChart(Component component) {
        Chart chart = new Chart(component);
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = .5 * Math.PI/(npoints-1);
```

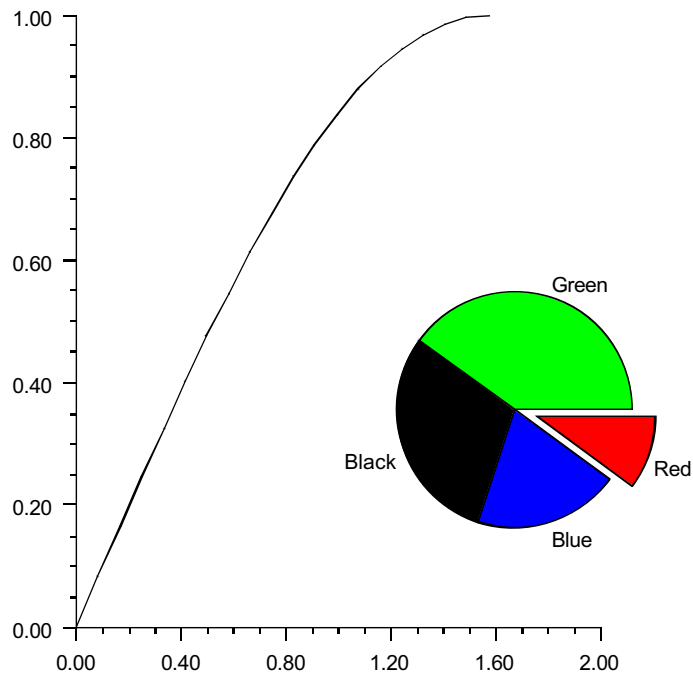
```
double x[] = new double[npoints];
double y[] = new double[npoints];

// Generate some data
for (int i = 0; i < npoints; i++){
    x[i] = i * dx;
    y[i] = Math.sin(x[i]);
}
new Data(axis, x, y);
return chart;
}
}
```



## Appendix C: Picture-in-Picture

The picture-in-picture effect can be obtained by using the Viewport attribute. This sets the fraction of the screen into which the chart is to be drawn. The Viewport attribute's value is a `double[ 4 ]` containing  $\{xmin, xmax, ymin, ymax\}$ , on a  $[0,1]$  by  $[0,1]$  grid with  $(0,0)$  at the top-left.



This chart tree for the above picture consists of a chart with an AxisXY child and a Pie child. (The Pie class is a subclass of Axis.) The viewport attribute of the Pie node is set to a non-default value.

```
import com.imsl.chart.*;
import java.awt.Color;

public class SamplePnP extends JFrameChart {

    public SamplePnP() {
        Chart chart = getChart();
        createLineChart(chart);
        createPieChart(chart);
    }

    private void createLineChart(Chart chart) {
        AxisXY axis = new AxisXY(chart);

        int npoints = 20;
        double dx = 0.5 * Math.PI/(npoints-1);
        double x[] = new double[npoints];
        double y[] = new double[npoints];
        for (int i = 0; i < npoints; i++){
            x[i] = i * dx;
            y[i] = Math.sin(x[i]);
        }
        new Data(axis, x, y);
    }

    private void createPieChart(Chart chart) {
        double y[] = {10., 20., 30., 40.};
        Pie pie = new Pie(chart, y);
        pie.setLabelType(pie.LABEL_TYPE_TITLE);
        pie.setViewport(0.5, 0.9, 0.3, 0.8);

        PieSlice[] slice = pie.getPieSlice();

        slice[0].setTitle("Red");
        slice[0].setFillColor(Color.red);
        slice[0].setExplode(.2);

        slice[1].setTitle("Blue");
        slice[1].setFillColor(Color.blue);
    }
}
```

```
        slice[2].setTitle("Black");
        slice[2].setFillColor(Color.black);

        slice[3].setTitle("Green");
        slice[3].setFillColor(Color.green);
    }

    public static void main(String argv[]) {
        new SamplePnP().setVisible(true);
    }
}
```





# Index

## A

- annotation 85
- Apache 131
- applications, adding charts to 8
- area plots 18
- array XML tags 135
- automatic scaling 89
- axis
  - crossed 117
  - custom tranformation of 110
  - labels 91
  - layout 88
  - limits 88
  - setting 87
  - tick marks 88, 104
  - titles 99
  - units 103

## B

- background 119
  - color 119
  - gradient 121
  - pattern 123
- bar charts 53
  - grouped 55
  - legends for 59
  - simple 53
  - spacing bars 61
  - stacked and grouped 57
  - width of bars in 61
- bitmaps 191
- box plots 46

## C

- candlestick charts 40
- chaining method calls 6
- charts
  - bar 53
    - simple 53
    - spacing 61
    - stacked and grouped 57
    - width 61
  - contour 62
  - grouped bar 55
  - histogram 64
  - labels for 82
  - legends for 59
  - pie 44
  - titles for 81
  - see *also* plots
- color
  - background 119
  - gradient 121
  - text 80
- contour 62
- custom axis transformation 110

## D

- dashed lines 72
- data point labels 82
- date/time
  - weekends 94
- dates in labels 92
- dates, transforming 89
- density, of tick marks 88
- dotted lines 72

## E

- enumerators, in XML 137
- error bar plots 30
- Extensible Markup Language
  - see XML

## F

- fills 75
  - colored 75
  - gradient 75
  - images for 78
  - outline colors for 75
  - outlines for 75
  - patterned 78
  - tiled 78
  - types of 75
- fonts 79
- formatting text 80
- function plot 22

## G

- gradient background color 121
- gradient fills 75
- graphs
  - function 22
  - spline 24
- grid lines 108
- grouped bar charts 55

## H

- help popups 129
- high-low-close plots 36
- histograms 64
- horizontal error bar plots 32

## I

- image files 191
- ImageIO
  - bitmap 191
- implicit nodes 4

## J

- JAI
  - see Java Advanced Imaging

- JAI API
  - bitmap 192
- JAR files 189
- Java Advanced Imaging
  - using for image files 191
  - using for servlets 183
- Javadoc 1

## L

- labels 82
  - dates in 92
  - for axes 91
  - string 97
- legends 59, 125
- line
  - color 72
  - dashed 72
  - dotted 72
  - plots 14
  - width 72
- line and marker plots 16
- lines 72
  - grid 108
  - see also line 72
- log plot 26
- loglog plot 28

## M

- marker
  - color 74
  - patterns 74
  - plots 16
  - size 74
  - thickness 74
  - type 73
- markers 73
- method calls, chaining 6
- mixed error bar plots 34
- mouse selection 169
- multiple axes 114

## N

- normal distribution histograms 64

## P

- pattern
  - background 123
  - fills 78
- picking 169
- picture-in-picture 197
- pie charts 44
- plots
  - area 18
  - box 46
  - candlestick 40
  - error bar 30
  - function 22
  - high-low-close 36
  - horizontal error bar 32
  - labels for 82
  - line 14
  - line and marker 16
  - log 26
  - loglog 28
  - mixer error bar 34
  - polar 67
  - scatter 10
  - semilog 26
  - titles for 81
  - vertical error bar 30
  - see *also* charts
- PNG files 193
- polar plots 67
- popups 129
- printing 178

## R

- random values 12
- reference lines 21
- reference manual 1

## S

- Scalable Vector Graphics (SVG)
  - bitmap 194
- scaling
  - automatic 89
  - input values 89
  - output values 90

- scatter plots 10
- scientific notation 91
- semilog plot 26
- serialization 180
- server
  - applications 183
  - directories 189
- servlets 183
- solid background 119
- spline graph 24
- stacked bar charts 57
- stock-price charts
  - candlestick 40
  - high-low-close 36

## T

- text 79
  - angle 80
  - color 80
  - fonts 79
  - formatting 80
  - size 79
  - style 79
  - typeface 79
- tick mark density 88
- tick marks 104
- titles
  - axis 99
  - plot 81
- tool tips 129
- transformation
  - axes 110
- transformations
  - dates 89
  - XML 140

## U

- units, in axes 103

## V

- vertical error bar plots 30
- viewport attribute 197

## **W**

- web servers [183](#)
- weekends
  - skipping [94](#)

## **X**

- XML [131](#)
  - array tags [135](#)
  - attribute tags [134](#)
  - enumerated types [137](#)
  - parser [131](#)
  - requirements [131](#)
  - syntax [131](#)
  - transformations [140](#)
  - XSLT [140](#)
- XSLT [140](#)

## **Z**

- zooming [172](#)